Improving Understanding and Trust with Intelligibility in Context-Aware Applications

Brian Y. Lim

May 2012
CMU-HCII-12-104

Human-Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Thesis Committee:
Anind K. Dey (Chair), Carnegie Mellon University
Scott E. Hudson, Carnegie Mellon University
Aniket Kittur, Carnegie Mellon University
Margaret M. Burnett, Oregon State University

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

This work was supported by the National Science Foundation under grant 0746428 and the author's National Science Scholarship (PhD) from the Agency for Science Technology And Research, Singapore. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author and do not necessarily reflect those of the funding agencies.

Keywords: intelligibility, explanations, context-awareness, ubiquitous computing (ubicomp), human-computer interaction.

ABSTRACT

To facilitate everyday activities, context-aware applications use sensors to detect what is happening and use increasingly complex mechanisms (*e.g.,* by using big rule-sets or machine learning) to infer the user's context and intent. For example, a mobile application can recognize that the user is in a conversation and suppress any incoming calls. When the application works well, this implicit sensing and complex inference remain invisible. However, when it behaves inappropriately or unexpectedly, users may not understand its behavior. This can lead users to mistrust, misuse, or even abandon it. To counter this lack of understanding and loss of trust, context-aware applications should be *intelligible*, capable of explaining their behavior.

We investigate providing intelligibility in context-aware applications and evaluate its usefulness to improve user understanding and trust in context-aware applications. Specifically, this thesis supports intelligibility in context-aware applications through the provision of explanations that answer different question types, such as: *Why* did it do X? *Why* did it *not* do Y? *What if* I did W, What will it do? *How* can I get the application *to* do Y?

This thesis takes a three-pronged approach to investigating intelligibility by (i) *eliciting* the user requirements for intelligibility, to identify what explanation types end-users are interested in asking context-aware applications, (ii) *supporting* the development of intelligible context-aware applications with a software toolkit and the design of these applications with design and usability recommendations, and (iii) *evaluating* the impact of intelligibility on user understanding and trust under various situations and application reliability, and measuring how users use an interactive intelligible prototype. We show that users are willing to use well-designed intelligibility features, and this can improve user understanding and trust in the adaptive behavior of context-aware applications.

ACKNOWLEDGEMENTS

I would like to thank my advisor Anind Dey for his time, support, wisdom, and guidance. His patience has allowed me to pursue my research interests and his comments have helped to me seek the big picture in my research and identify implications for my findings. Likewise, I thank my thesis committee members — Scott Hudson, Niki Kittur, and Margaret Burnett — to guide my research and strengthen its technical contributions and experimental rigor.

Though the pursuit of a PhD may feel lonely at times, I am fortunate to have many friends and colleagues supporting me, and keeping me company along the way.

To my office buddies of four years — Tawanna Dillahunt and Min Kyung Lee — for the fond memories and much commemoration and commiseration of our research and PhD lives, thank you. To my academic brother, Matthew Lee, for your listening ear and warm friendship, thank you.

To many students, faculty, and staff at CMU, the HCII and the Ubicomp Lab for your invaluable discussions, time, feedback expertise, and help — Vincent Aleven, Daniel Avrahami, Maria Brooks, Kerry Chang, Laura Dabbish, Scott Davidoff, Christian Köhler, Chloe Fan, Denzil Ferreira, Charles Gouin-Vallerand, Chris Harrison, Eiji Hayashi, Jin-Hyuk Hong, Gary Hsieh, Kevin Huang, Ruogu Kang, Sara Kiesler, SeungJun Kim, Sunyoung Kim, Rebecca King, Queenie Kravitz, Andy Ko, Bertha Lam, Ian Li, Somchaya Liemchetcharat, Bryan Low, Jen Mankoff, Gabriala Marcu, Noboru Matsuda, Andreas Möller, Stephen Oney, Bryan Pendleton, Stephanie Rosenthal, Carolyn Rosé, John Santerre, Aubrey Schick, Julia Schwarz, Choonsung Shin, Jessica Stanley, Kanupriya Tavri, Leonghwee Teo, Karen Tang, Eran Toch, Dezhong Yao, Zhiquan Yeo, Haiyi Zhu, Brian Ziebart — thank you.

To my friends from PCCO OIF and the Abundant Life group for welcoming me into their community and forging meaningful relationships, thank you.

To my parents for your immeasurable sacrifices and selfless love, for nurturing me in my formative years, and your continued support, thank you.

To my love, Huilin, for your many *jia you*'s to encourage me, nudging me through my work, and supporting me through many moments. Thank you for always being here for me.

TABLE OF CONTENTS

FIGURES

TABLES

1 INTRODUCTION

Over the past 20 years, with the miniaturization and commoditization of computing power, we have moved away from the *desktop paradigm* of computing to that of *ubiquitous* computing (Ubicomp). This manifests Weiser's vision of a world with ubiquitous, invisible computing [Weiser, 1991] embedded in smart ambient environments and carried by end-users in small devices. Anticipating, adapting, and servicing user needs, these Ubicomp systems were envisioned to work calmly and quietly, remaining in the background [Weiser and Brown, 1997], not getting in the way of the users' work or activities.

An important part of this Ubicomp vision is *context-aware* computing [Dey, Abowd, and Salber, 2001; Schilit, Adams, and Want, 1994] with applications that automatically adapt and tailor their behavior in response to the user's current situation (or *context*), such as the user's activity, location, and environmental conditions. Using sensors to recognize or infer the user's intent or situation, context-aware applications do not need *explicit* user input to carry out their functions. Hence, these applications *implicitly* determine what is happening and complement the user's activity without needing the user's attention. Examples of context-aware applications include:

- Mobile tour guides, *e.g.*, CyberGuide [Abowd *et al*., 1997], GUIDE [Cheverst *et al*., 2000]),
- Reminder systems, *e.g.*, CybReminder [Dey and Abowd, 2000];
- Monitoring and awareness systems, *e.g.*, Digital Family Portrait [Mynatt *et al*., 2001], embedded assessment of the elderly [Lee and Dey, 2010; 2011], domestic activity [van Kasteren *et al*., 2008], coworker awareness [Lim, Brdiczka, and Bellotti, 2010];
- Interruption management, *e.g.*, for Instant Messaging [Avrahami and Hudson, 2006], on the mobile phone [Lim and Dey, 2011a; Rosenthal, Dey, and Veloso, 2011], and in the office [Tullio *et al*., 2007];
- Coordination, *e.g.*, family transportation [Davidoff *et al.*, 2011];
- Service or device automation, e.g., Intelligent Office System [Cheverst et al., 2005]

Consider using context-awareness to manage interruption on a mobile phone. With the proliferation of smart mobile phones, mobile applications can leverage embedded sensors in the phones to provide context-awareness. A compelling application is for the phone to automatically detect what the user is doing to determine whether it is an appropriate time for the user's contacts to call and interrupt her. For example, the application can detect if the user is in a conversation (using the microphone for sensing and machine learning for inference) at the office (using Wi-Fi or GPS sensing), or detect if she is driving a car (using the accelerometer for sensing and machine learning for inference). Using a set of rules, it can infer whether the user is available.

In the previous example, as with many context-aware applications, the user does not need to explicitly inform the application of her availability, or more generally, of her contextual situation, and can expect the application to serve her need to be uninterrupted without her involvement. The application uses **implicit sensing**, and **complex inference** to support context-awareness. However, these designs and capabilities can lead to some user interaction issues.

1.1 THE PROBLEM — LACK OF INTELLIGIBILITY

Since context-aware applications sense implicitly and act quietly, these applications lack the affordances [Gibson, 1979] to allow end-users to be aware of what they know or what they are doing. Bellotti *et al*. [2002] point out that with the vision of Ubicomp making the interface invisible, it would become difficult for these systems to manifest themselves and allow users to make sense of them. Dourish [1996] argues that interactive systems should give "accounts" — reflective representations of their operations and *externally observable states*.

The complex inference mechanisms employed by context-aware applications also increase the difficulty of understanding how these applications reason and decide. Bellotti and Edwards [2001] propose that context-aware systems must be *intelligible* — *"able to represent to their users what they know, how they know it, and what they are doing about it."* They believe that, along with enforcing user *accountability*, intelligibility "must be present for context-aware systems to be useable, predictable, and safe." Bellotti *et al*. [2002] also challenge Ubicomp systems to support *alignment* between the user and system, by making the system state *perceivable*, *persistent*, and *query-able*, and providing timely and appropriate feedback. Indeed, this lack of intelligibility has been empirically observed. Barkhuus and Dey [2003a, b] found that although end-users want to use context-aware applications, they have serious issues with the lack of understandability, loss of control, loss of privacy, information overload; users find automatic behavior useful but difficult to understand.

Trust in automation guides reliance when the complexity of the automation makes a complete understanding impractical [Lee and See, 2004]. This lack of system intelligibility in context-aware applications and user confusion can lead users to mistrust and misuse, and even abandon them [Muir, 1994; Muir and Moray, 1996]. Therefore, ensuring end-users have sufficient user *trust* of these systems is crucial to supporting their adoption. Lee and See [2004] described three attributes of trust in automation: predictability, performance, and purpose. *Predictability* and *performance* are particularly relevant to the problem of the lack of intelligibility. Without sufficient understanding of context-aware applications, end-users will find these applications' behaviors less predictable, and this can compromise user trust. Furthermore, context-aware applications are prone to ambiguity and uncertainty [Greenberg, 2001]. This can cause them to make wrong inferences and misbehave, compromising their performance. A common strategy for improving the performance of context-aware applications involves user mediation, where the user resolves uncertainty [Dey *et al*., 2002]. Nevertheless, without intelligibility, end-users will struggle to determine the causes for uncertainty and may not be able to improve the system performance.

1.2 A SOLUTION — EXPLANATIONS FOR INTELLIGIBILITY

Providing explanations is a popular way to improve user understanding and user trust [Johnson, 1993] in Intelligent Systems. Dzindolet *et al* [2003] found that even though users lose trust in intelligent decision aids which make occasional errors, providing a description of why the aid might fail can help to increase users' trust. Explanations have been shown to improve user understanding and performance in expert systems (*e.g.*, knowledge-base systems [Davis, Buchanan, and Shortcliffe, 1977, Gregor and Benbasat, 1999], intelligent decision aids [Glass, McGuinness, and Wolverton, 2008; Haynes, Cohen, and Ritter, 2009]) and end-user systems (*e.g.*, recommender systems [Herlocker, Konstan, and Riedl, 2000], intelligent user interfaces [Myers *et al.*, 2009]).

We employ the same strategy of providing users with explanations of application state, inference logic, and behavior for context-aware applications. For example, a context-aware application may mis-infer the user's availability to receive phone calls, and allow a colleague to call him at the library. Intelligibility will allow the user to learn why this apparent mistake happened. It could tell him that the application correctly inferred his location at the library, but that he had forgotten to set a rule to be unavailable, or that his colleague ignored social norms and called anyway.

1.2.1 THESIS STATEMENT

In this thesis, we explore how to provide intelligibility in context-aware applications through explanation interfaces. We aim to support both *developers* to design and implement intelligible context-aware applications, and evaluate the benefits and limitations of intelligibility on *end-users*. With the intelligibility explanations we develop in this thesis, we claim that:

> ***Intelligibility*** *in context-aware applications can improve end-users'* ***understanding*** *of how these applications work and, consequently, increase end-users'* ***trust*** *to use these applications.*

1.2.2 THESIS APPROACH

To prove this thesis statement, we approach the problem in three high-level stages. First, we (i) explore what intelligibility is and define it through exploratory work, then we (ii) facilitate and support intelligibility so that it is easier to provide it, and finally, (iii) we evaluate the usefulness of intelligibility towards the thesis goals. Figure 1.1 outlines the chapters in this dissertation.

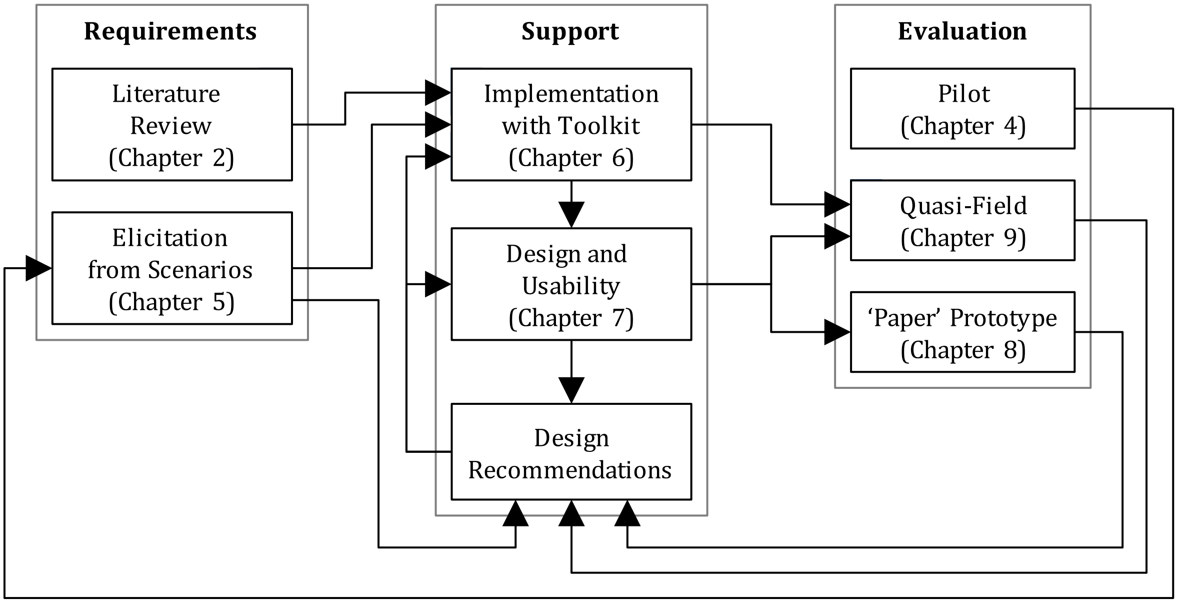

Requirements
Literature Review (Chapter 2)
Elicitation from Scenarios (Chapter 5)
Support
Implementation with Toolkit (Chapter 6)
Design and Usability (Chapter 7)
Design Recommendations
Evaluation
Pilot (Chapter 4)
Quasi-Field (Chapter 9)
'Paper' Prototype (Chapter 8)

Figure 1.1. Three-stage approach to thesis with various projects connected by progression. Arrows indicate how findings and implications from one study applies to the next. We summarize our taxonomy for Intelligibility in Chapter 3.

I) REQUIREMENTS GATHERING AND SPECIFICATION

In the first stage, we sought to define a framework for intelligibility. We accomplish this with a literature review of explanations in intelligent systems (Chapter 2), and empirical work eliciting what explanations potential users of context-aware applications would like to know (Chapter 5). To this end, we have defined a taxonomy of explanation question types.

II) FACILITATION, SUPPORT, AND GUIDELINES

The next stage implements the requirements as determined from the taxonomy of intelligibility, and provides generalized support for implementing intelligibility in context-aware applications through a software toolkit and design recommendations. We facilitate the implementation of intelligibility with the Intelligibility Toolkit (Chapter 6), and also explored and evaluated design and usability issues to derive guidelines for providing and presenting intelligibility (Chapter 7).

III) EVALUATION

In the final stage, we evaluate intelligibility in context-aware applications. Using the toolkit and design guidelines, we can rapidly prototype intelligibility in context-aware applications to test our hypotheses. We investigated the impacts of different explanation types on user understanding and trust of context-aware intelligent systems (Chapter 4). Next, through questionnaires, we evaluated the impact of intelligibility on user impression of context-aware applications that are uncertain or certain of their inferences (Chapter 8). We followed this with an evaluation of an interactive prototype of an intelligible context-aware mobile application, where we investigated the extent of usage of intelligibility, how well or poorly users understood the application inferences, and their perceived usefulness of the explanations (Chapter 9).

1.2.3 INTELLIGIBILITY AS EXPLANATION TYPES

We support intelligibility through an explanation query paradigm (*e.g.*, [Wick and Slagle, 1989; Ko and Myers, 2003], where users can obtain explanations to questions about the context-aware applications, such as:

1. **What** is the current value of the context?
2. **Certainty**: how certain or confident is the application of this inference?
3. **Why** is this context the current value X?
4. **Why Not**: why isn't this context value Y, instead?

5. **How To**: when would this context take value Y?
6. **What if** the conditions are different, what would this context be?

Categorizing explanations into these Explanation Types allows us to systematically investigate their usefulness and how to support their provision in context-aware applications. We detail our taxonomy of Explanation Types in Chapter 3.

1.3 SCOPE AND DEFINITIONS

There are several terms and concepts that are central to this dissertation and we define them here.

In this thesis, we focus on providing and evaluating explanations in context-aware applications used by lay end-users for everyday computing activities. We use the definition of **context-awareness** as defined in [Dey, Abowd, Salber, 2001; Schilit, Adams, and Want, 1994] regarding a *positivist*, *constructionist* view of understanding of the environment and the user through constituent contextual cues and signals that are sensed, aggregated, interpreted, and inferred. These can include sensors around the house (*e.g.*, thermostats, brightness sensors), in computer software (*e.g.*, keyboard and mouse activity), worn on the body or in mobile devices (*e.g.*, accelerometers, microphones); and inferred activities and intentions such as domestic activity (*e.g.*, making breakfast, using the toilet), and mobile availability and activity (*e.g.*, driving, talking in a meeting). On the other hand, the use of intelligibility, especially in a social application (*e.g.*, Laκsa in Chapter 7), can support the *interactionist*, *phenomenological* view of context [Dourish, 1994], where context is relational, dynamic, depends on the social interactions, arises from activity, and is co-constructed with the user. Intelligibility can provide users with more information to make better sense of the situation.

There can be many different types of users of intelligent systems, with different relationships to the systems and different domain expertise. We have scoped our investigation into context-aware applications to cover **"everyday" activities** as defined in [Abowd, Mynatt, an Rodden, 2002; Greenfield, 2006] (*e.g.*, reminder systems, interruption management), rather than work task-oriented or professional decision aids (*e.g.*, medical diagnosis knowledge bases, task planning). Finally, we target **lay end-users** as the consumers of the intelligibility features we seek to provide. We do not expect these users to have technical or computer science expertise, nor will they necessarily have deep interest in understanding the detailed operation of novel context-aware applications. Instead, we expect these users to primarily focus on their activities and pay attention to intelligibility occasionally, *e.g.*, when the applications misbehave or act unexpectedly.

We intend for context-aware applications to provide intelligibility to help end-users **learn** and **understand** them. Much research has been performed on explanations in intelligent systems, using different terms to describe an intelligible application, such as: explainable, interpretable [Mozina *et al*., 2004], transparent [Cheverst *et al*., 2005; Cramer *et al*., 2008; Höök, 2000], scrutable [Assad et al., 2007; Barua, Kay, and Kummerfeld, 2011], palpable [Rimassa, Greenwood, and Calisti, 2005], "glass box" [Höök *et al*., 1996], white-box [Herlocker, Konstan, and Riedl, 2000], seamful [Chalmers and MacColl, 2003], *etc*. Given the complex inference mechanisms and sensors used in context-aware applications, there will be terms and concepts central to their operation that end-users may not understand. Therefore, intelligibility can help end-users to learn the relevant *terminology* and *concepts*, so that they may properly scaffold and form more accurate *mental models* [Johnson-Laird, 1983]. We do not intend for end-users to learn these concepts to the extent which students learn from their coursework (as is the intention of Intelligent Tutoring Systems, *e.g.*, [Anderson *et al*., 1995]), nor do we expect end-users to understand the application to be able to debug their code (*e.g.*, Whyline [Ko and Myers, 2003]). We aim to use intelligibility to allow end-users to understand the factors or sensors that influence the inference and decision making in context-aware applications, so that they may be *aware* of and *appreciate* the competence of the applications' complex inference (assuming reliable performance). We also want end-users to understand the *limitations* of the applications.

We aim is to improve end-user **trust** by improving the end-user's "ability to estimate predictability of the [application's] behaviors" by making the behaviors "observable" [Muir, 1994]. Lee and See [2004] identified three processes underlying trust: analytic, analogical, and affective. Analogical trust is influenced by the context, environment of use, and other social factors such as reputation. Affective trust is influenced by the user's emotional response and allows her to reduce her cognitive burden when deciding how much to trust the application. Parasuraman and Miller [2004] found that differences in machine etiquette (*e.g.*, providing messages at appropriate or disturbing times, whether polite or impolite) can influence user trust more than the automation reliability. This demonstrates an influence of affect on user trust. **Analytic trust** relates to the user's understanding of the logic of the application and is influenced by the user's cognition. Though we acknowledge the importance of each type of trust, in this thesis, we focus on promoting analytic trust by improving the user's understanding of the application's behavior. We also aim to help users to better calibrate their trust

[Dzindolet *et al.*, 2003] in context-aware applications with their increased understanding of the competence and limitations of these applications.

Finally, Edwards, Newman, and Poole [2010] noted that low-level infrastructure on which applications are built should also be made intelligible. Although we provide a toolkit to support intelligibility, our focus in this thesis is to support intelligibility for end-user applications.

1.4 CONTRIBUTIONS

This dissertation makes a number of major contributions:

- Evidence that end-users want intelligibility in context-aware applications.
- A taxonomy of explanation types that end-users desire to have provided for context-aware applications.
- A toolkit for supporting the development of intelligibility in context-aware applications.
- Algorithms to generate multiple explanation types from several rules and machine learning inference models.
- Design recommendations for intelligibility features.
- A prototype of an intelligible context-aware application developed through several iterations.
- Investigation of caveats and limitations of providing intelligibility (usability issues and intelligibility of uncertain systems)
- Evidence that end-users can use intelligibility features to learn about context-aware inferences and behaviors
- Evidence that providing intelligibility can improve end-user understanding and trust in context-aware applications

1.5 OUTLINE

The rest of the dissertation is organized as follows:

To give a background to this dissertation, in Chapter 2, we review explanations in intelligent systems, various taxonomies of explanations, and systems that provide explanations to users. In Chapter 3, we give an overview of intelligibility as defined in this dissertation. We describe research questions that drove various projects in the thesis and introduce a taxonomy of explanation types that intelligible context-aware applications can provide. The following chapters are organized chronologically and in the order that follows from the chain of reasoning in our research questions.

Chapter 4 describes early work demonstrating the usefulness of intelligibility to help end-users understand and trust the output of a context-aware intelligent system. Particularly, we compare the effectiveness among four explanation types. Subsequently, in Chapter 5, we describe our expansion of the list of explanation types through an elicitation study by presenting questionnaires of various applications and scenarios to participants.

Chapter 6 describes how we support the implementation of our taxonomy of explanation types with an Intelligibility Toolkit to automatically generate and present explanations from multiple inference models. However, the toolkit does not provide design recommendations on how to present explanations to users. In Chapter 7, we describe a user study that explored design and usability issues for intelligibility interfaces in a context-aware application prototype, Laκsa.

Having designed a usable, intelligible context-aware application, we evaluate the impact of intelligibility. Chapter 8 describes a questionnaire study that investigated the positive and negative impact of intelligibility for application inferences with high or low certainty, respectively. Chapter 9 describes a quasi-field study evaluating the usage and usefulness of intelligibility in our Laκsa prototype, showing how usage of intelligibility helps end-users to better understand and troubleshoot the application inference.

In Chapter 10, we conclude the dissertation with a summary of its contributions and a discussion of its limitations. We include several appendices describing detailed technical aspects of the Intelligibility Toolkit, descriptions of the intelligibility user interface of the Laκsa prototype, and experiment study materials.

2 Related Work: Explanations in Intelligent Systems

In this chapter, we review the explanation taxonomies developed in several research domains of different types of intelligent systems. Research in several domains have explored the impact of explanations to improve user trust and acceptance of intelligent systems, including knowledge-based systems (see a review in [Gregor and Benbasat, 1999]), task processing systems (*e.g.*, [Glass, McGuinness, and Wolverton, 2008; Haynes, Cohen, and Ritter, 2009; McGuinness *et al*., 2007; Silveira, de Souza, and Barbosa, 2001]), intelligent tutoring systems (*e.g.*, [Graesser, Person, and Huber, 1992; Graesser, Baggett, and Williams, 1996]), recommender systems (*e.g.*, [Herlocker, Konstan, and Riedl, 2000; Cramer *et al*., 2008]), case-base reasoning (CBR) (*e.g.*, [Kofod-Petersen, Cassens, and Aamodt, 2008; Sørmo, Cassens, and Aamodt, 2005]), end-user debugging (*e.g.*, [Ko and Myers, 2004; 2009; Myers *et al*., 2006]), and context-aware systems (*e.g.*, [Assad *et al*., 2007; Cheverst *et al*., 2005; Tullio *et al*., 2007; Vermeulen *et al*., 2009]), *etc*. These domains can be categorized into two groups, namely, expert systems handling professional tasks and end-user systems handling "everyday" activities. We discuss how we draw inspiration from these works that have investigated explanations over the past several decades, and identify gaps and opportunities for providing explanations for context-aware applications in ubiquitous computing (Ubicomp).

2.1 Explanations in Expert Systems

Much early research on explanations in intelligent systems were focused on expert systems to help professionals to learn how the system makes decisions, or to help novices to learn about decision making. As such, several frameworks of explanations have been developed.

2.1.1 KNOWLEDGE-BASED SYSTEMS

Drawing from explanation facilities of many knowledge-based systems (KBS), Gregor and Benbasat [1999] identify three classification methods of explanation type: **content**, **presentation** format, and **provision** mechanism. They found that KBS systems provide four content types of explanations:

1. **Trace or line of reasoning.** In response to the typical "why" question, this explanation type describes the decision processes taken by the system, *why* or *how* it came to its result. Explanations that EMYCIN [Van Melle, Shortliffe, and Buchanan, 1984] provided are of this type.
2. **Justification or support.** Introduced in the XPLAIN system [Swartout, 1983], this type of explanation provides deeper domain knowledge to justify the system's process. These deep explanations can incorporate different types of knowledge such as analogies, cases, and text books.
3. **Control or strategic.** Introduced in NEOMYCIN [Clancey, 1983], this type of explanation explains the "system's control behavior, and problem solving strategy." This provides the user with the design rationale that the developers employed for the application logic.
4. **Terminological.** Distinguished by Swartout and Smoliar [1987], this type of explanation familiarizes users with domain terms and concepts by providing terminologies and definitions.

There are several factors, such as **user expertise**, that affect when certain explanation content types are more important. For example, novice users would use justification and terminology explanation types more as they learn how to use the expert system; expert users would mainly use explanations to resolve anomalies and for verification, so they would prefer reasoning traces and control types of explanations.

Presentation styles used in KBS systems have been identified to fall into two categories: **Text-based** and **Multimedia**. Text-based explanations can either be in the form of programming language syntax, a canned text of the programming logic, or natural language translations of the logic. Multimedia explanations use graphics, images, animations, or sound.

Gregor and Benbasat have also identified three types of mechanisms to provide explanations: **user-invoked**, **automatic**, and **intelligent**. User-invoked (also known as on-demand, optional, or voluntary) explanations can be provided through menus, commands, and hyperlinks, and users can choose whether or when to invoke them. Automatic explanations are provided all the time, and users do not get a choice of whether to receive them. To maximize exposure of certain explanations, and minimize the perceived effort of obtaining these explanations, Everett [1994] recommends making these explanations automatic. Intelligent provision of explanations depend on the system determining when is most appropriate to provide the explanations. Gregor and Benbasat discuss employing user modeling to track their expertise and mental model (and whether they are making mistakes) for the system to determine when to provide explanations.

2.1.2 INTELLIGENT DECISION AID

The knowledge-based systems discussed by Gregor and Benbasat [1999] deal mainly with supporting decisions, or helping users decide what to do, rather than acting on their behalf. On the other hand, there is a growing number of systems that are being designed to be more proactive, and have greater autonomy to carry out tasks. These systems, also called *intelligent agents*, would have to gain the trust of users before they can be widely accepted. One way to increase user trust is to increase transparency in these systems, such as by answering explanation questions. Haynes, Cohen, and Ritter [2009] did an extensive review of explanations in intelligent agents (systems that "make use a knowledge-base and algorithm to carry out its responsibilities"), using a wider scope of systems than just KBS. They extend and reorganize Graesser *et al.*'s [1992] classification of 13 explanation-seeking questions into a framework of four main explanation types: ontological, mechanical and operational explanations, and design rationale.

- **Ontological** explanations provide "what" information to help users make sense of a concept or a component of the system, including:
    - **What – identity.** Basic ontological information about the existence of an agent or agent component, or its identifier.
    - **What – definition.** Information beyond simply identifying an agent or component and involves providing it with some meaning in context through definitions.
    - **What – relation.** Information about the static structural relation between agents or their components, such as spatial information.
    - **What – event.** Especially distinguished, this is information about entities that are primitives in describing causal explanations, and can provide temporal information.

- **Mechanistic** explanations deal with the *how* of agent behavior. The main type of question is "How does it work?" This type of explanations provides information about how different components interact to give rise to more complex actions.
- **Operational** explanations answer the question of "How do I (the user) use it (the system)?" They provide instructions for the user or other agents to enact some agent behavior.
- **Design rationale** explanations deal with *why* questions at multiple levels from system component constraints to designer intentions to law-like relations. In relation to the taxonomy provided by Gregor and Benbasat, the design rationale spans reasoning trace and strategic. Haynes *et al*. categorize design rationale into four parts:

    - **Deductive-Nomological (D-N).** Explanations referring to some law or law-like relation between entities and/or agents. This is based on the D-N model that suggests that explanations should take the form of deductive statements predicated on well-established truths [Hempel, 1965].
    - **Functional.** Design intent of the function of a created agent or component.
    - **Structural.** Explanations that refer to the structure of the system constraints that cause an entity or event to happen.
    - **Pragmatic.** Explanations to requests that depend on the user's interest value. These explanations are in response to either *why not* or *what if* questions.

In an empirical study using a virtual pilot cognitive model intelligent agent, Haynes *et al*. found that most explanation seeking questions (58%) were ontological, followed by mechanistic (19%), then operational (12%) and design rationale (11%).

McGuinness and colleagues have explored explanation needs for task processing systems, particularly with the Cognitive Assistant that Learns and Organizes (CALO, 2007). Focusing on *temporal* characteristics, McGuinness *et al*. [2007] articulated several types of explanation questions that users of task processing systems are interested in:

- **Motivation for tasks.** In response to the question "why are you doing <task>?", answer strategies can (i) include identifying the task requestor (attribution), (ii) indicating that the task is a subtask that supertask depends on, (iii) indicating the task is next-in-step of a task procedure, and (iv) indicating that certain terminating conditions have not yet been met.

- **Task status.** This regards to (i) what tasks are being done, (ii) what the status of those tasks are, (iii) whether certain tasks are not being done (*what didn't*), and (iv) whether any tasks are being hindered.
- **Task history.** This regards to (i) what the system has done recently, (ii) what it has started recently, (iii) why it *did* a task (in the past, as opposed to why it *is doing*), (iv) why it *didn't* do a task, (v) how it did a task, and (vi) and variants of reasoning regarding what didn't questions.
- **Task plans.** While task history looked into past actions, task plans looks into the future planned actions. This regards to (i) what the system will do next, (ii) when it will start the task, (iii) why, and (iv) how it expects to do it.
- **Task ordering.** This regards to (i) why a task is being done before another, (ii) why some other task has not yet been started, and (iii) what needs to be done to complete a task.
- **Explicit time questions.** This regards to (i) when a task will begin, or (ii) end, (iii) when a task happened, (iv) how long it took to complete, (v) why a task took so long to complete, (vi) why a task is already being done instead of later.

While users of task processing systems may have many questions regarding time, they have other information requirements before they can appropriately trust these applications. Through structured interviews with users of CALO, Glass *et al*. [2008] investigated several factors that influence their level of trust. They used Silveira *et al*.'s taxomony [2001] of users' frequent doubts to derive a list of question types users are interested in:

- **Choice**: What can I do right now?
- **Procedural**: How can I do this?
- **Informative**: What kinds of tasks can I accomplish?
- **Interpretive**: What is happening now? Why?
- **Guidance**: What should I do now?
- **History**: What have I already done?
- **Descriptive**: What does this do?
- **Investigative**: Did I miss anything?
- **Navigational**: Where am I?

These questions are ordered by the rated importance from the interviews. While question types defined by McGuinness *et al*. [2007] were mainly about time, and about the system, these questions are about the user and his activity.

2.1.3 INTELLIGENT TUTORING SYSTEMS

While not quite expert systems to aid workers in their work, Intelligent Tutoring Systems provide expert knowledge (of the domain or concept being studied) to students. The knowledge or information can be provided via explanations. Graesser *et al*. have explored how students ask questions and derived several explanation types and reasons for question asking. Graesser and McMahen [1993] four conditions when questions are asked:

- **Anomalous event.** Questions are asked about the causes and consequences of an unusual event, *e.g.*, if someone faints in a restaurant.
- **Contradiction.** Questions are asked to resolve a contradiction between two propositions, *e.g.*, two people who claim to be married but are not wearing wedding rings.
- **Obstacle to a goal.** Questions are asked to remove or circumvent an obstacle to a goal, *e.g.*, when a car fails to start, the driver will ask why it will not start and how it can be fixed.
- **Equally attractive alternatives.** Questions are asked to break a tie between a set of alternatives, *e.g.*, pros and cons of switching jobs, choosing different products.

From empirical analyses of questions in educational settings, Graesser and Person [1994] grouped Lehnert's [1987] 16 question categories into three depth levels:

- Simple / shallow questions
    - **Verification:** invites a yes or no answer
    - **Disjunctive:** Is *X*, *Y*, or *Z* the case?
    - **Concept completion:** Who? What? When? Where?
    - **Example:** What is an example of *X*?
- Intermediate questions
    - **Feature specification:** What are the properties of *X*?
    - **Quantification:** How much? How many?
    - **Definition:** What does *X* mean?
    - **Comparison:** How is *X* similar to *Y*?

- Complex / deep questions
    - **Interpretation:** What does *X* mean?
    - **Causal antecedent:** Why / How did *X* occur?
    - **Cause consequence:** What next? What If?
    - **Goal orientation:** Why did an agent do *X*?
    - **Instrumental / procedural:** How did an agent do *X*?
    - **Enablement:** What enabled *X* to occur?
    - **Expectation:** Why didn't *X* occur?
    - **Judgmental:** What do you think of *X*?

While these questions are not specifically for end-users to ask of automated systems, many of them are relevant (*e.g.*, example, feature specification, comparison, causal antecedent, goal orientation, expectation). Point and Query, an educational software [Graesser, Langston and Baggett 1993] provides explanations to questions in terms of levels of knowledge:

- **Taxonomic knowledge:** What does *X* mean? What are the types of *X*? What are the properties of *X*?
- **Sensory knowledge:** What does *X* look like? What does *X* sound like?
- **Goal-oriented procedural knowledge:** How does a person use / play *X*?
- **Causal knowledge:** What causes *X*? What are the consequences of *X*? How does *X* affect sound? How does a person create *X*?

2.1.4 RELATION TO CONTEXT-AWARE APPLICATIONS

The aforementioned frameworks provide a rich design space for different types of explanations. However, they cater to expert systems with users who carry out tasks that require expert decision making. Context-aware applications in ubiquitous computing focus on helping lay end-users in "everyday" activities [Abowd, Mynatt, and Rodden, 2002], so their users would require a different set of explanations. For example, we expect the functional purpose of context-aware applications to be clearer than expert systems because, as everyday products, their functional scope would be limited. Therefore, we do not anticipate functional explanation types to be very necessary. Nevertheless, some of these explanation types remain useful for context-aware applications.

In this thesis, the explanations we provide for intelligibility are mainly about the application's *line of reasoning*, or *mechanistic*. We treat context-aware applications as inference and decision agents, and, through intelligibility, reveal their reasoning process. We take a user-centered approach, and therefore, also provide *pragmatic* design rationale explanations to explain to end-users how the application inferred in the context of the user's goals (why not) or present understanding of the situation (what if). While users should not have to be overly bothered by technical terminology when using everyday applications, to explain some of the low-lying contexts and reasoning traces, *terminological* explanations may be needed to help users learn relevant explanatory concepts. We also expect users to act on the information they learn from intelligibility, but they would need to know how they can modify or control the context-aware application. Therefore, *operational* explanations would also be relevant to provide in context-aware applications.

2.2 EXPLANATIONS IN END-USER SYSTEMS

Research into explanations for KBS or task processing systems tends to focus on trained or reasonably knowledgeable users. However, explanations can be useful for novice end-users to understand unfamiliar programs too, even those that help with their everyday tasks.

2.2.1 RECOMMENDER SYSTEMS

Currently, explanations of end-user systems are most accessible to people through online recommender systems like Amazon's recommendation of products, Pandora.com's song selection, *etc*. Herlocker, Konstan, and Riedl [2000] described two sources of errors: model/process, and data.

- **Model/process errors** are due to the limited feature space of the computational model used;
- **Data errors** are due to (i) not enough data, (ii) poor or bad data, or (iii) high variance data.

To support explanations, Herlocker et al. discuss white-box and black-box models. The **white-box model** divides the Automated Collaborative Filtering (ACF) system into three parts: user *profile* ratings, *similarity measures* used to compare profiles, and the model or *mechanism* of how the ratings are combined to form recommendations. These explanation capabilities may help users understand the conceptual model of the system, but this may not be desirable all the time, especially for guarding proprietary methods. The **black-box model** is appropriate for such situations, and use alternative information to explain the system. Techniques include providing information about *past performance justification* (e.g. that the system was 80% correct in the past when recommending this), and using *external supporting evidence* (justification type explanations).

Tintarev [2007] classifies the explanation types used in recommender systems in several types such as **case-based**, **content-based**, **collaborative**, **demographic**, and **knowledge-based**. Much of these explain the recommendations regarding the *similarity* of the attributes of the entities of interest (*e.g.*, speed of camera), of the user (*e.g.*, demographic information), preference similarities between users (*e.g.*, the user preferring low prices).

To explore the impact of explanations on consumers' trusting beliefs in online shopping (e-commerce) recommendation agents (RAs), Wang and Benbasat [2007] examined the effects of three types of explanations:

- **How** explanation to reveal the *line of reasoning* used by the RA. This increased perceived *benevolence* that the RA acts in the consumer's interest.
- **Why** explanation to *justify* the importance and purpose of the RA to consumers. This increased perceived *competence* (performance) and *benevolence* in the RA.
- **Trade-off** explanation to offer *objective* decision guidance to help consumers identify differences in features between products. This increased perceived *integrity* that the RA adheres to a set of principles (*e.g.*, honesty, justice, objectivity).

Note their use of the terms *why* and *how* differ from how they are used in the rest of this dissertation.

Cramer *et al*. [2008a, b] investigated the effects of transparency in an art recommender, Cultural Heritage Information Personalisation (CHIP) system, on user trust. They considered three versions of CHIP: non-transparent, transparent (provides **Why** explanations listing properties the current recommendation shares with artworks the user had previously rated positively, and 'sure' (showing a **Confidence** rating of the system's recommendation). They found that providing Why explanations increased user *acceptance* of the system, but did not improve user *trust*. Furthermore, they found that Confidence (Certainty) explanations did not improve acceptance or trust.

Even though these similarity-based approaches are highly effective for recommender systems, context-aware applications also use context information about the physical environment and situation. Moreover, context-aware applications can use other types of models to make inferences. From a literature survey of context-aware applications [Lim and Dey, 2010] and in Section 6.2, we found that the most popular models are indeed different: rules, decision trees, and naïve Bayes classifiers. Therefore, while explanations have been richly studied for recommender systems, research into explanations for context-aware applications remains an open problem.

2.2.2 CASE-BASE REASONING

Given the focus on unique and similar products or entities that recommender systems have, recommender systems can also be considered as systems operating on a collection of cases. This lends itself nicely to applying techniques in Case-Based Reasoning (CBR). For example, Top Case [McSherry, 2005] provides explanations to discriminate between different cases and explain ***why*** one is better than another. It explains in terms of attributes of the cases, indicating whether they are the same or different for different cases, and which attributes do not affect the recommendation.

Some research has sought to provide frameworks for explanations in CBR. Roth-Berghofer [2004] describes five explanation types of [Spieker, 1991] relevant to CBR:

- **Conceptual** explanations to describe the meaning of concepts
- **Why** explanations to describe the *cause* or *justifications* for an event
- **How** explanations as a special case of Why explanations to describe the causal chain of the decision process
- **Purpose** explanations to describe the purpose of a fact or object
- **Cognitive** explanations as a special case of Why explanations. The previous four explanation types explain the physical world in which the CBR system operates on, while these explain the processing and behavior of the system.

Roth-Berghofer describes knowledge containers (vocabulary, similarity measures, adaptation knowledge, and case-base) as components of the CBR system which contribute variously to these explanations.

Sørmo, F., Cassens, J., and Aamodt [2005] identified five goals for explanations in CBR to satisfy:

- **Transparency** to explain how the system reached the answer
- **Justification** to explain why the answer is a good one
- **Relevance** to explain why a strategy is relevant
- **Conceptualization** to clarify the meaning of concepts and vocabulary
- **Learning** to teach the user about the domain

Cassens [2008] employ problem frames [Jackson, 2000] to model explanation *machines* and *system knowledge* to meet these goals.

CBR has also been applied to ambient intelligent systems (*e.g.*, [Cassens and Kofod-Petersen, 2007; Kofod-Petersen and Aamodt, 2003; Ma *et al.*, 2005; Zimmermann, 2003]). For example, Cassens and Kofod-Petersen [2007], added explanation capabilities the CREEK architecture [Aamodt, 2004] in a simulated hospital ward domain. For user-centric explanations, they distinguish between *context-awareness* (inferring the situation) and *context-sensitivity* (acting according to the situation) and respectively provide different explanations:

- **Elucidate** why the system identifies a particular situation (context-awareness). This explanation exposes the system's assumptions of the environment to *justify* what it believes.
- **Explicate** why a certain behavior was taken (context-sensitivity). This explanation points out the *relevance* of the system performing a particular action.

2.2.3 END-USER PROGRAMMING

End-user programming considers users whose primary task is not to program the application, but who still do so to facilitate their task or configure the application. For example, people who use spreadsheets to tabulate and calculate budgets can be considered end-user programmers. Ko and Myers [2005] found that end-user programmers of the Alice programming environment [Conway *et al.*, 2000] asked questions when their expectations are unmet. They asked **why did** questions when something unexpected occurs and **why didn't** questions when something expected does not happen. Ko and Myers subsequently develop the Whyline system [2004, 2009] that traverses the program tree to generate reasoning traces within the program code to generate *why did* and *why didn't* explanations:

- **Why did** the program do X?
- **Why didn't** the program do Y?

Kulesza *et al*. [2011] developed the What You See is What You Test for Machine Learning (WYSIWYT/ML) method that supports *systematic testing* of machine learning applications, particularly for high criticality tasks. WYSIWYT/ML provides explanations of

- **Confidence** to indicate how certain the system was of its classification
- **Similarity** of how different the example is from previously trained data

- **Relevance** of how able the system is to perform the classification
- **History** to help users track inference changes after the users make edits

This is complementary to our approach of supporting *ad hoc* testing of context-aware applications, where end-users serendipitously learn about the applications' behavior. It assumes that some end-users will take the effort to perform such a rigorous test. We do not assume such enthusiasm and effort of end-users, and explicitly measure their usage in our study described in Chapter 9. As demonstrated with WYSIWYT/ML [Shinsel *et al*., 2011], explanation and testing facilities can also be helpful for multiple stakeholders or "mini-crowds" that share the use of intelligent agents to collectively improve the behavior of a machine learning system. However, we focus on single-user or single-viewer use of intelligibility in this thesis.

Although machine learning is becoming popular for developers of intelligent adaptive systems, it still remains difficult for developers to understand and debug their programs. Patel *et al*. has investigated the *classification pipeline* [Patel *et al*., 2008], and developed several tools (*e.g.*, Gestalt [Patel *et al*., 2010], Prospect [Patel *et al*., 2011]) to help developers implement classifiers and analyze their data. Although the applications investigated were for end-users, Patel *et al*. focused on supporting programmers familiar with machine learning. We focus on end-users with no knowledge of machine learning in this thesis.

2.2.4 INTELLIGENT AND ADAPTIVE USER INTERFACES

Intelligent and adaptive user interfaces are closely linked to context-aware, but typically describe desktop-based applications, *e.g.*, spam filters, email sorters, or office application assistants. They typically perform user modeling to understand the user needs and adapt accordingly. To increase their predictability to end-users, Höök [2000] argues for user-adaptive systems to be *transparent*. She describes three *glass box* levels from [Brown, 1989]:

- **Domain transparency** for the user to see the application domain or concepts relevant to the system,
- **Internal transparency** for the user to see the internal workings of the system, and
- **Embedding transparency** for the user to see a whole picture of how she relates to the system.

Myers *et al.* [2006] apply the Whyline explanation types (**why did** and **why didn't**) to end-user "everyday" productivity tools with the Crystal framework to support these explanations in a sample text editor that has auto-correct features. Following this question-asking approach, Kulesza *et al*. [2009] investigated the provision of **why…** and **why not…** explanations for an email client that uses the naïve Bayes machine learning classifier to sort email. Due to the probabilistic nature (rather than deterministic or rule-based) of the naïve Bayes classifier, reasoning traces were not used for the explanations, but a representation of weights from various inputs (keywords). Explanations were provided as a rich visualization of bar charts.

Kulesza *et al*. [2012] explored whether end-users can quickly build and recall sound structural mental models of an intelligent music recommender system. They found that scaffolding with a human tutor can help end-users to build mental models with greater soundness, and allow them to subsequently better operate the system. Even though the scaffolding was not done through the system interface, this gives evidence that end-users can learn to better and effectively understand such complex systems. In this thesis, we minimize scaffolding via human tutors or instructions, such that end-users learn about the system behavior and inference through the intelligibility provided via the interface.

2.2.5 RELATION TO CONTEXT-AWARE APPLICATIONS

It is intuitive that end-users would also ask why and why didn't questions for other "everyday" applications, and, in the proposed thesis, we take this approach of providing explanations to these questions, but generalize it for context-aware applications. Our work leverages some explanation techniques from Kulesza *et al*., extending them to explain physical contexts that are more relevant for context-aware applications. Furthermore, the overall approach in end-user programming is to allow the end-user to debug the application when it behaves inappropriately. We broaden the use of explanations to be used in more situations, even when the application is functioning appropriately.

2.2.6 UBIQUITOUS AND CONTEXT-AWARE COMPUTING

Context-aware applications for ubiquitous computing present new challenges for providing explanations to end-users. These applications would penetrate everyday life and have a wide impact on end-users [Abowd, Mynatt, and Rodden, 2002]. Furthermore, many of these systems would automatically gather information (contexts) about the user and environment and implicitly take various actions [Dey, Abowd, and Salber, 2001]. However, such activity done “quietly” without the user’s knowledge [Weiser and Brown, 1997], without much transparency, can be disconcerting to users who may like to know how their information is being used.

Bellotti and Edwards [2001] state that context-aware applications must be intelligible: being able to "represent to their users *what* they know, *how* they know it, and what they are *doing* about it." They proposed a framework for intelligibility and accountability including four principles:

1. Inform the user of current contextual system *capabilities* and *understandings*.
2. Provide ***feedback*** including:
   - **Feedforward**: What will happen if I do this?
   - **Confirmation**: What am I doing and what have I done?
3. Enforce ***identity*** and ***action disclosure*** particularly with sharing restricted information: Who is that, what are they doing, and what have they done?
4. Provide ***control*** (and defer) to the user, over system and other user actions that impact her, especially in cases of conflicts of interest.

In this thesis, we cover aspects of the first two principles exposing the application capabilities by selecting relevant information and informing users of the systems' understandings through generating explanations. We also support feedback through various explanation types.

2.2.6.1 INTELLIGIBLE CONTEXT-AWARE APPLICATIONS

A simple form of intelligibility is to show the **Certainty** of the application's inference. Antifakos and colleagues showed that uncertainty improved task performance speed of participants when certainty is high [2004], and that participants verified automatic settings made by a context-aware system less often when its certainty was high or medium [2005]. In studies of presenting location information [Dearman *et al*., 2007; Lemelson *et al*., 2008], visualizations of location certainty were found to improve user performance with location-based services.

Some early *intelligible* context-aware applications provide end-users with a modest amount of explanations to give them insight mainly by providing *transparency* (showing the application's **underlying state**) and traceability (showing **reasoning trace**) information. Cheverst *et al*. [2005] investigated how much users would want to know about rules governing a context-aware system and whether to control it. The system takes actions depending on context changes (and history) and the user model (e.g. preferences), and displays to users its **rules** of a fuzzy decision tree and its **certainty** about the inference. McCreath, Kay, and Crawford [2006] explored the difference in *scrutability* of different machine learning classifiers (sender identity, keywords, TF-IDF, decision trees, naïve Bayes) in their Intelligent-Electronic Mail Sorter. The Daily Activities Diarist [Metaxas *et* *al*., 2007], an awareness display to support aging in place (like the Digital Family Portrait [Mynatt *et al*., 2001]), employs *narratives* complemented with *graphical visualizations* to provide **semantic cues** and explanations. Tullio *et al*.'s interruptibility displays [2007] explain how they determine a manager's interruptibility by exposing the values of sensors in the manager's room. Panoramic [Welbourne *et al*., 2010] provides **reasoning trace**, **location** status, and **history** explanations to explain location events through a visualization of parallel timelines of sensed and rule-determined events. Vermeulen *et al*. explored several interfaces to provide intelligibility in ambient intelligent (AmI) environments. They projected **trajectory** visualizations along the wall of an AmI room, tracing the application operation from sensor input (*e.g.*, camera motion sensor) to actuator output (*e.g.*, room light) [Vermeulen *et al*., 2009]. The PervasiveCrystal [Vermeulen *et al*., 2010] also explains for processes in a smart environment by providing **Why** and **Why Not** explanations from a mobile screen display.

2.2.6.2 FRAMEWORKS TO SUPPORT INTELLIGIBILITY IN CONTEXT-AWARE COMPUTING

Some frameworks and toolkits have also been developed to provide wider support for intelligibility in context-aware applications. SpeakEasy [Newman *et al*., 2002] supports querying and displaying of the **states** of devices (PCs, printers, projectors, *etc*.) in an environment, allowing users to discover if they are available, they have failed, *etc*. PersonisAD [Assad *et al*., 2007] defines a distributed framework to support explanations by resolving **identities** and **associations** of devices, locations, people, *etc*. It makes user models *scrutable* so that users can control which parts of their user model can be private or public and visible to the sensing environment. Personis-LF [Barua, Kay, and Kummerfeld, 2011] extends this concept of scrutability to life-long personalization and adds capabilities to control *forgetting* information. While this is important for deployed systems, this thesis does not cover the scope of longitudinal use of intelligibility. Hardian *et al*. [Hardian, 2006; Hardian, Indulska, and Henricksen, 2008] added a *Logging and Feedback Layer* along with a *Query Interface* to the Pervasive Autonomic Context-aware Environments (PACE) middleware [Henricksen and Indulska, 2006] to reveal elements that influence application behavior. However, as pointed out by Fong [2010], these components expose information that is too low-level and overly technical.

Dey and Newberger [2009] provide the Enactors toolkit to support intelligibility and control in context-aware applications by adding the Enactor component to the Context Toolkit. For intelligibility, it allows applications to provide **input context values**, and **reasoning traces**. For control, it exposes parameters that the UI layer of the application can allow users to interact with and manipulate. This thesis extends the scope of intelligibility to allow users to ask more questions of the application's state and inference mechanism. For example, users would be able to ask about an anomaly with a Why Not question, and ask about a possible future scenario with a What If question.

Vermeulen [2010] proposed to explore the design space for providing and presenting intelligibility in Ubicomp systems along the dimensions of:

- **timing** — before, during, or after an event
- **generality** — general, or domain-specific
- **degree of co-location** — whether intelligibility is provided in the same UI or separately
- **initiative** — user, or system initiated
- **modality** — visual, auditory, haptic
- **level of control** — not controllable to fully programmable

This thesis takes a different approach to investigate intelligibility in context-aware applications. Rather than explore multiple presentation styles for intelligibility, we have explored the provision of intelligibility from an *information-centric* perspective. End-users are considered information *consumers* of explanations, and intelligible applications as information *providers* through the explanations they can generate, and present. Presentation styles are definitely important for the effective assimilation of explanations and conveyance of intelligible information, but we have treated finding the best solutions for presenting explanations in different applications mainly as a design exercise.

Inspired by our taxonomy of explanation types (see Chapters 4 and 6), TOSExp (TinyOS Explained) [Bucur, 2011] supports intelligibility in embedded context-aware applications by providing *static* explanations to explain the **Inputs** values and **Outputs** range of the application, and **What If** and **How To** explanations that describe hypothetical behaviors of the application. It operates at an embedded systems level to provide bit-accurate explanations that while being very precise, may suffer from a lack of user-friendliness by being too low level or too detailed. This thesis focuses on systems and applications at higher programming abstraction layers (*i.e.*, application logic) and also prioritizes explanations that are more usable for end-users.

Targeting end-user preference models for context-aware systems, Fong *et al*. [2010, 2011] developed an intelligible preference modeling approach that expresses preferences in terms of if-then-else rules.

Their system can generate explanations to questions of **What**, **Why**, **Why Not**, **How To**, and **Control**. As such, this is limited to preference modeling and rules. In this thesis, we do not restrict our contributions to just rules and include machine learning models and models for other purposes, such as activity recognition.

Metaxas [2010] investigated supporting intelligibility in the Contextual Range Editor (CoRE) for end-users to configure rules for awareness systems. He consider rules presented in text templates and whether to present the rules in **disjunctive normal form** (DNF) or **conjunctive normal form** (CNF) depending on the *affinity* of logical terms (*e.g.*, "driving" and "running" have higher affinity than "running" and "talking"). In Chapter 6, we also consider DNF for representing explanations of rules, and can integrate Metaxas' findings within the framework of the Intelligibility Toolkit.

2.2.6.3 INTERPRETABLE MACHINE LEARNING

Machine learning is a popular technique to enable inference and activity recognition in many context-aware applications (see review in Chapter 6). For example, machine learning is used to recognize what activity an occupant in the home is performing [van Kasteren *et al.*, 2008]. To support intelligibility in these applications using machine learning models, these inference models will need to be intelligible too. Indeed, much work in the artificial intelligence and machine learning computing community have sought to make these models *interpretable*. In this thesis, we focus on explanations for the inference process rather than the learning or training process.

Some learned models are trivial to explain (*e.g.*, decision trees that can be transformed into rules) by just traversing through the program branches to provide reasoning traces. Some learned models, in particular additive classifiers (*e.g.*, Naïve Bayes, linear Support Vector Machine (SVM), and Linear Regression), are less intuitive, but still relatively easy to make interpretable (*e.g.*, Mineset [Brunk *et al*. 1997], Nomograms [Mozina *et al.*, 2004]; ExplainD [Poulin *et al.*, 2006]). These explanation methods present visualizations to users and indicate decision processes based on weights placed on different features. There also remain several "black-box" classifiers (such as Artificial Neural Networks) that are not directly interpretable. One way to try to make them reasonably interpretable is by using case-base reasoning to provide an alternative explanation [Nugent and Cunningham, 2005], and another way is to extract rules from them [Núñez, Angulo, and Català, 2002; Tickle *et al.*, 1998].

2.3 SUMMARY

In summary, much research investigating the provision of explanations in intelligent systems have demonstrated a positive impact on user understanding and trust. Research in the domain of context-aware computing is also nascent and has shown some promise, but more work is required to provide stronger support for intelligibility and gain better insight about how intelligibility impacts users. This thesis proposes to deepen this research, and provide concrete contributions towards providing intelligibility in context-aware applications. In the Chapter 3, we describe how the nature of context-aware applications pose research questions for providing intelligibility, and describe the taxonomy of explanations we investigated to answer these questions in the thesis.

3 EXPLANATION TYPES FOR INTELLIGIBILITY

In Chapter 2, we reviewed the different types of explanations provided in various intelligent systems. In this chapter, we introduce the research questions that have driven our investigation and then describe the taxonomy of intelligibility explanation types we have developed to make context-aware applications intelligible.

As mentioned in the earlier section, context-aware applications use *implicit sensing*, and *intelligent inference* to determine the user's context so as to perform appropriate actions. For Ubicomp systems, context-aware applications have been primarily developed to support everyday activities, such as tracking the user's physical activity to monitor her exercise, recognizing activity in the home to provide timely medical assistance, determining her availability to others, providing recommendations based on where she is and what she is doing, reminding her to pick up the milk when she is located at the grocery store, *etc*. They sense implicitly to minimize obtrusiveness and interruption to the user; they automatically sense the situation rather than requiring the user to manually tell them what is happening. Context-aware applications are increasingly using sophisticated inference mechanisms due to the growing complexity of contexts they need to understand, particularly for activity recognition. For inference, they use big rule sets and machine learning algorithms to handle diverse situations, and to be more robust to exceptional cases. All these improve the accuracy in properly and calmly understanding the user's context.

Unfortunately, these two factors of implicit sensing and intelligent inference also make context-aware applications difficult for end-users to understand. This is particularly problematic when the applications behave inappropriately or unexpectedly. In such cases, context-aware applications no longer remain invisible to the user's experience; instead, they become a puzzle. The users become frustrated if they cannot understand what has happened and why the application behaved unexpectedly. Eventually, this **lack of understanding** would lead to a **loss in trust** in the system's inference and behavior, and the eventual abandonment of them. Without a proper understanding of how context-aware applications work, users may also not be able to effectively control them to improve their performance for subsequent situations. Therefore, it is crucial for context-aware applications to be *intelligible*, so that they can explain what they sense and how they are inferring about the users' contexts.

3.1 RESEARCH QUESTIONS FOR INTELLIGIBILITY

Starting with a broad idea of intelligibility from Bellotti and Edwards [2001], we defined intelligibility for a context-aware application as the ability to answer or explain *questions* that users could ask. Given the implicit actions that context-aware applications take, end-users may not know what the application is doing, let alone assess whether it has performed appropriately. Hence, it is important for applications to make their action state explicit and provide *feedback* of what they are doing. This is supported by providing an explanation or answer to the question:

1. **What** is the current value of the context?

Continuing with the user-centric perspective of answering intuitive questions, we draw from the question-answering approach of the Whyline [Ko and Myers, 2004, 2009], with just *why* and *why not* questions. One can easily imagine a confused, exasperated, or inquisitive user asking the following questions:

2. **Why** is this context the current value X?
3. **Why Not**: why isn’t this context value Y, instead?

Why asks what factors caused or influenced the inference outcome, and Why Not asks why an alternative inference was not made. In a similar manner as the Whyline, we answer these questions by providing *mechanistic* explanations that specifically describe the inference over the instance the end-user is asking about. Note that we do not enforce a particular structure of explanations to answer these questions. They could be answered with *rule traces* (line of reasoning) or some other structures. We do not explain these in terms of *design rationale* or *purpose*, which relate to the underlying assumptions, concepts, or objectives driving how the application behaves.

As an extension of Why and Why Not questions, end-users may want to ask questions relating to the general rules or model under which the application makes inferences. This can allow the users to generalize their understanding of how the application works to better *predict* future behavior. Specifically, we provide explanations for the questions:

4. **How To**: when would this context take value Y?
5. **What if** the conditions are different, what would this context be?

How To explanations are a generalization of Why explanations, but they do not specifically target any instance. In terms of rule traces, this explanation type can be expressed by listing all traces that achieve the desired inference. What If explanations support the *feedforward* type of feedback, where end-users can investigate what the application will do in a future or hypothetical scenario.

We began our investigation of providing intelligibility in context-aware applications with this initial set of five explanation types. This thesis aims to show that intelligibility can improve user understanding and trust of context-aware applications. We would especially like to show this with the scope of intelligibility that we have defined based on multiple question types. Specifically, our first investigation sought to answer the research question:

RQ1. DOES INTELLIGIBILITY HELP USERS IMPROVE THEIR UNDERSTANDING AND TRUST OF CONTEXT-AWARE INTELLIGENT SYSTEMS?

Even though this has been proven true with narrower forms of intelligibility (transparency, scrutability, *etc.*) in related work, we explored how supporting the various question types independently affect user understanding and trust in context-aware applications. Our work, presented in Chapter 4, shows that providing some explanation types (Why and Why Not) are more effective than others in improving user understanding and trust.

These successful results from our first study showed that providing intelligibility is a promising avenue for research. Next, we sought to carefully explore the scope of questions that users would ask of context-aware applications. Specifically:

RQ2. WHAT ARE THE INTELLIGIBILITY NEEDS OF END-USERS IN CONTEXT-AWARE APPLICATIONS?

Answering this question will help to ensure that the intelligibility we aim to provide will be relevant to users and can better satisfy their informational needs. In work presented in Chapter 5, we conducted user-centered, empirical research to elicit what information users wanted to know of context-aware applications, when the applications behaved under various situations. We identified more explanation types, and expanded our taxonomy of explanation types.

To improve end-users awareness of what the application knows, much previous work in adaptive or context-aware applications have investigated the principle of making the application *transparent*. One way to support transparency is to fully reveal the internal input state of the application. This answers the question:

6. **Inputs**: what *factors* and *values* affect this context?

One could distinguish between naming the input *sources*, and the *value* taken by each input at the time of interest. Users are also interested in the range and diversity of actions or responses that context-aware applications. Considering an application model as an input-output functional model, this supports the explanation for the question:

7. **Outputs**: what other values can this context take?

Given the ambiguity and uncertainty in sensing and inference, context-aware applications are not necessarily deterministic in their decision logic. Hence, users are also interested in asking:

8. **Certainty**: how confident is inference of this value?

With increased knowledge and understanding of the applications, users will also want to be able to reconfigure or control the application to improve its behavior. This asks the question:

9. **Control:** how can I control the application to improve it?

Finally, we determined some circumstances in which users asked for information additional to what the context-aware application may model for its function. For example, wanting to see a video capture of the room where an elderly family member was detected to have fallen. Providing this extra information helps answer the question:

10. **Situation:** what else is happening in this situation (not about the application, but about the circumstance)?

Similarly, users want to know if the application has taken other actions meanwhile:

11. **What Else:** what else did the application do?

With the study described in Chapter 5, we identified which explanation types users ask of context-aware applications. However, it remains difficult for application developers to implement intelligibility in context-aware applications, especially with such a wide range of explanation types. This brings us to the next research question:

RQ3. HOW CAN WE SUPPORT THE IMPLEMENTATION OF INTELLIGIBILITY IN CONTEXT-AWARE APPLICATIONS?

We chose to provide toolkit support for developers to easily add intelligibility to their context-aware applications (Chapter 6). We developed the Intelligibility Toolkit that provides extensible components to support the automatic generation of explanations, and mechanisms to process the explanation information into simpler forms that end-users may easily interpret. However, this technical contribution did not provide final solutions to how the explanations should be presented to end-users. This leaves unaddressed the next research question:

RQ4. HOW CAN WE DESIGN INTELLIGIBILITY FOR CONTEXT-AWARE APPLICATION TO BE USABLE FOR END-USERS?

We answer this question with a think-aloud usability study described in Chapter 7, where we designed Laκsa, a complex context-aware application that uses multiple input contexts and various rules and machine learning classifiers. This application was implemented as an interactive prototype for participants to engage with. In this study, we explored several design principles for intelligibility, and evaluated how users interpret explanations from an intelligible context-aware application. Our findings provide insights and design recommendations for providing usable intelligibility in context-aware applications.

We considered context-aware applications with inference models that infer a certainty distribution over multiple Outcomes. Instead of a single What value, there can be a non-zero Certainty of inferring each of the possible Output values. We support and later manifest this as an aggregation of explanations Outputs + Certainties. An alternative point of view is that the What explanation is extended to include a range of output values.

12. **Outputs + Certainties**: how confident is inference of all possible values?

As we investigate providing explanations with a real-world interactive prototype, new explanation types become more relevant and important, namely:

13. **When**: when was the context inferred as this value?
14. **History**: what was the inference at an earlier time, T? Why did it make that inference at time T? *Etc*.

Historical explanations can help to provide users with a *confirmation* of what they and the application have done in the past. Furthermore, explanations about history include not just the inferred value at that time, but also any other event-dependent explanations about the event.

As context-aware applications begin to use esoteric sensors and features for inference, we also include textual descriptive information to help end-users to learn the *terminology* used by the application and key concepts.

15. **Description**: what is the meaning of the context terms and values?

Description explanations can also be used to *justify* the behavior of the application by describing the implications of various context values, and describe the *rationale* for the application to consider various features or inference mechanisms.

At this stage, we investigated how to provide intelligibility through gathering requirements, providing technical support, and recommending design principles. This allows developers and designers to more easily and carefully implement, provide, and present intelligibility in context-aware applications. This also enables us to explore our hypotheses on the impact of intelligibility with more realistic intelligible context-aware applications. Logically, we next address research questions relevant to evaluation in light of realistic issues. One concern is that context-aware applications are not always certain of what they infer, and providing intelligibility may not be helpful when they are uncertain. This could be because users learn about the applications' weaknesses. This brings up the research question:

RQ5. WHEN IS INTELLIGIBILITY HELPFUL AND HARMFUL FOR CONTEXT-AWARE APPLICATIONS WITH DIFFERENT CERTAINTIES?

We conducted a large online controlled study with a between-subjects experiment design to investigate the interaction effect of providing intelligibility and of application certainty on user impression of two context-aware applications. This is described in Chapter 8. We found that above a threshold of about 80% certainty, providing intelligibility improves user impression of the application performance. However, below that threshold, providing intelligibility harms user impression because it reveals the weaknesses of the application.

This result deepens our earlier findings in Chapter 4, and considers nuances in the impact of intelligibility in context-aware applications. At this point, much of our work on evaluating intelligibility has focused on questionnaire studies and 'paper' prototypes of realistic albeit fictitious context-aware applications. With the Laκsa prototype (Chapter 7), we sought to increase realism in investigating intelligibility with an interactive prototype. However, intelligibility was shown "always on" to participants, so they were biased to look at the explanations. This brings forward the question:

RQ6. EVEN IF INTELLIGIBILITY CAN IMPROVE USER UNDERSTANDING AND TRUST, WILL USERS WANT TO USE IT, AND, IF SO, HOW MUCH?

We address this question with the study described in Chapter 9. Using a quasi-field experiment with four scenarios, we let participants freely use a fully interactive intelligible context-aware application on a mobile phone. We logged their usage of the intelligibility features, and interviewed participants to evaluate their understanding of the application behavior. We found that participants do use intelligibility without prompting, and that more extensive and deeper usage helps them to better understand the application behavior.

3.2 TAXONOMY OF INTELLIGIBILITY EXPLANATION TYPES

We have introduced several explanation types in the previous section, and in our empirical study in Chapter 5. Here, we summarize these into a framework of explanation types for intelligible context-aware applications.

Explanation Type		**Question**	**Explanation**
What (Output Value)	Top Value	What is the inferred value?	Shows the value of the inferred output.
	Outputs	What are the inferred values?	Lists multiple other likely alternative values.
	What Else	What else (other actions) did the application do?	Informs what other actions the application is simultaneously doing.
Certainty	Top	What is the confidence of inferring the current value *X*?	Shows the Certainty of inference.
	Certainties	What is the confidence of inferring all possible values?	May include certainties of inferring other values.
When		When was value *X* inferred?	Indicates the time that the inference was made.
Why		Why was value *X* inferred?	*With the Intelligibility Toolkit, this explanation can be provided as a Rule Trace or as Weights of Evidence.* Describes the triggered rule(s) or weights of evidence for the inference.
Why Not		Why was value *Y* not inferred?	*Same format as Why.* Describes the un-triggered rules or difference in weights of evidence for why an alternative value *Y* was not inferred.
Input Values		What are the factor values / What is the input state?	Describes the values of all input factors.
Situation		What else is happening with the situation? What is the ground truth?	Provides a description or playback of the recorded ground truth to convey a richer picture or experience of the situation. *E.g.*, showing a video of the sensed scene, providing an audio recording of the sound recognition source.
History*		*Provides the same range of explanations, but for a historical event or inference at a specific time in the past.	

Table 3.1. Dynamic instance-based explanation types explaining the inference of a specific event. These explanations will differ for every instance the application acts.

Explanation Type		Question	Explanation
What If		What will be the inferred value, if the input values are *W*?	Provides a hypothetical What or What Else answer given user-queried input values. Requires user input to specify / constrain some input values.
How To		How can I get the application to infer *Y*?	*Similar format as Why, but* Explains in terms of an alternative output value *Y*, instead of *X*.
How To If		How can I get the application to infer *Y*, given a subset of input values *W*?	*Similar format as How To, but* Requires user input to specify / constrain some input values.
Control	Parameter Values	What parameters can I change to control the application behavior?	Describes how to control and adjust parameters or attributes to change the application behavior (*e.g.*, in a manner exposed in [Dey and Newberger, 2009]). *We do not cover this explanation type in this thesis*
	Rules / Model	What rules or settings can I change?	Describes how to add/edit rules or the model. *We do not cover this explanation type in this thesis.*

Table 3.2. Dynamic general explanation types explaining the inference model of the context-aware application.

Explanation Type		Question	Explanation
Inputs Factors		What factors / sources influence this inference?	Lists all input factors / sources for the application.
Outputs (Options)		What are the possible output values for this inference?	Lists all possible values or actions that the application may produce or perform.
Description	Terminology	What does this term mean?	Provides a textual description of a term or concept.
	Justification	What is the implication of this value?	Provides a textual description of the implication of a context value. *E.g.*, a high "Periods of Silence" in the sensed sound suggests talking noise because speech has more relative silence than voices.
	Rationale	What is the rationale for this inference?	Provides a textual description of the rationale of a process, rule, or inference mechanism. *E.g.*, the application considers sound activity when inferring availability because you may be in an impromptu meeting, and it detects your talking, even though your calendar is open (no events scheduled).

Table 3.3. Static general explanation types explaining the inference model of the context-aware application. For a static (fixed) model, these explanations will always be the same.

In the next chapters (4 to 9), we describe in detail the pieces of work that have been completed for this thesis.

4 INVESTIGATING THE INTELLIGIBILITY OF QUESTION TYPES

This chapter is an extension of the work presented in:

> Lim, B. Y., Dey, A. K., and Avrahami, D. (2009). Why and Why Not Explanations Improve the Intelligibility of Context-Aware Intelligent Systems. In *Proceedings of the 27th international Conference on Human Factors in Computing Systems (Boston, MA, USA, April 04 - 09, 2009). CHI '09*. ACM, New York, NY, 2119-2128.

This publication was a best paper honorable mention for a *CHI '09*.

ABSTRACT. Context-aware intelligent systems employ implicit inputs, and make decisions based on complex rules and machine learning models that are rarely clear to users. Such lack of system intelligibility can lead to loss of user trust, satisfaction and acceptance of these systems. However, automatically providing explanations about a system's decision process can help mitigate this problem. In this chapter, we present results from a controlled study with over 200 participants in which the effectiveness of different types of explanations was examined. Participants were shown examples of a system's operation along with various automatically generated explanations, and then tested on their understanding of the system. We show, for example, that explanations describing why the system *behaved* a certain way resulted in better understanding and stronger feelings of trust. Explanations describing why the system *did not behave* a certain way, resulted in lower understanding yet adequate performance. We discuss implications for the use of our findings in real-world context-aware applications.

4.1 INTRODUCTION

This chapter describes an investigation of a number of mechanisms for improving system intelligibility performed using several controlled online lab experiments. To investigate these intelligibility factors and their effects, we defined a model-based system representing a canonical *intelligent system* underlying a context-aware application, and an interface with which users could learn how the application works. We recruited 211 online participants to interact with our system, where each one received a different type of explanation of the system behavior. Our findings show that explaining *why* a system behaved a certain way, and explaining why a system did *not* behave in a different way provided most benefit in terms of objective understanding, and feelings of trust and understanding compared to other explanation types.

In this chapter, we first define a suite of intelligibility explanations derived from questions users may ask of a context-aware system and that can be automatically generated. We then describe an online lab study setup we developed to compare the effectiveness of these explanation types in a quick and scalable manner. Next we describe the experimental setup used to expose participants to our system with different types of intelligibility and the metrics we used to measure understanding, and users' perception of trust, and understanding. We present two experiments in which we investigated these factors, elaborating on the results and implications. We end with a discussion of all of our results and plans for future work.

4.2 INTELLIGIBILITY

Context-aware systems can confuse users in a number of ways. For example, such systems may not have familiar interfaces, and users may not understand or know what the system is doing or did. Furthermore, given that such systems are often based on a complex set of rules or machine learning models, users may not understand why the system acted the way it did. Similarly, a user may not understand why the system did not behave in a certain way if this alternative behavior was expected. Thus, our focus in the work presented here is on explanations that can be regarded as reasoning traces.

While a reasoning trace typically addresses the question of *why* and *how* the application did something, there are several other questions that end-users of novel systems may ask. We chose to following initial set of intuitive questions (adapted from [Dourish, Adler, and Smith, 1996]):

1. **What:** What did the system do?
2. **Why:** Why did the system do W?
3. **Why Not:** Why did the system not do X?
4. **What If:** What would the system do if Y happens?
5. **How To:** How can I get the system to do Z, given the current context?

Throughout this chapter we will refer to these as our five intelligibility question types, and the explanation addressing each of them as an explanation type.

Norman described two gulfs separating users' goals and information about system state [Norman, 1988]. Explanations that answer questions What, Why, and Why Not address the *gulf of evaluation* (the separation between the perceived functionality of the system and the user's intentions and expectations), while explanations answering questions What If and How To address the *gulf of execution* (the separation between what can be done with the system and the user's perception of that). With a partial conception of how a system works, users may want to know what would happen if there were some changes to the current inputs or conditions (What If). Similarly, given certain conditions or contexts, users may want to know what would have to change to achieve a desired outcome (How To).

This chapter deals with providing and comparing the value of explanations that address four of these intelligibility questions to investigate which of these explanations benefit users more. We label these explanation types: Why, Why Not, What If, and How To. Since the system we developed to evaluate the value of explanations, already explicitly shows the inputs and output of the system (see next Section on Intelligibility Testing Infrastructure), we did not investigate the What explanation.

4.2.1 HYPOTHESES

We hypothesize that different types of explanations would result in changes in users' user experience: *understanding* of the system and *perceptions* of trust and understanding of the system. We will now present our hypotheses about each of these intelligibility questions.

Why explanations will support users in tracing the causes of system behavior and should lead to a better understanding of this behavior. So, we expect:

H1: *Why* explanations will improve user experience over having no explanations (*None*).

Why Not explanations should have similar benefits to *Why* explanations; however, users' ability to apply *Why Not* explanations may not be as straightforward. There may be multiple reasons why a certain outcome did not happen; while a why explanation may be a single reasoning trace (or at least a small number of possible traces), a why not explanation is likely to contain multiple traces. Given this complexity, users will require more cognitive effort to understand how to apply the knowledge, and may do so poorly. As such, we expect:

H2: *Why Not* explanations will **(a)** improve user experience over having no explanations (*None*), but **(b)** will not perform as well as *Why* explanations.

Explanations for How To and What If questions would have to be interactive and dynamic, as they depend on example scenarios that users define themselves. Receiving these explanations should be better than receiving none at all. However, given that novice end-users are unlikely to be familiar with a novel system, they may choose poor examples to learn from, and learn less effectively than the Why explanations. So we expect:

H3: *How To* or *What If* explanations will **(a)** improve user experience over having no explanations (*None*), but **(b)** will not perform as well as *Why* explanations.

	Hypotheses	Experiment 1	Experiment 2
H1	None < Why	None < Why	None < Why
H2a	None < Why Not	None < Why Not	None < Why Not
H2b	Why Not < Why	None ≈ Why Not	None ≤ Why Not
H3a	None < (How To, What If)		None ≈ (How To, What If)
H3b	(How To, What If) < Why		(How To, What If) < Why

Table 4.1. Summary of hypotheses and results regarding the effect of explanation type on user experience (understanding and trust). '≈' means no significant difference (p=n.s.); '≤' means we hypothesize either a lower user experience or no difference.

To test these hypotheses (summarized in Table 4.1), we created a test-bed that allows simulating different types of intelligent systems and testing different explanation types. We describe this testing infrastructure next.

4.3 INTELLIGIBILITY TESTING PLATFORM

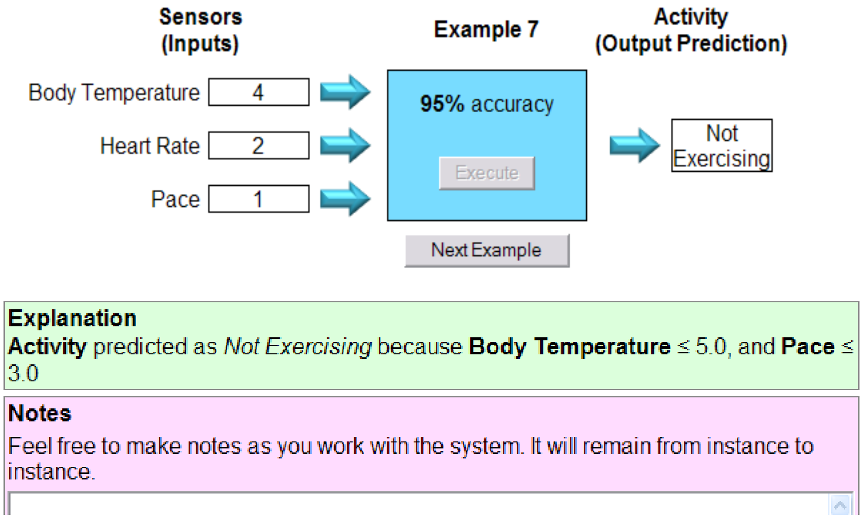
We developed a generalizable web interface that can be applied to various application domains to study the effect of the various mechanisms for providing intelligibility. Users interact with a schematic, functional *intelligible* system that could underlie a context-aware application: it accepts a set of inputs (*e.g.* Temperature, Humidity), and uses a model (for example, a decision-tree), to produce a single output (*e.g.*, Rain Likely, or Rain Unlikely). Users are shown different instances of inputs and outputs and can be given various forms of explanations (or no explanations) depending on what explanation type is being studied. To users who do not receive explanations, the system appears as a black box (only inputs and the output are visible).

This infrastructure allows us to efficiently and rapidly investigate different intelligibility factors in a controlled fashion and closely measure their effects; further, the online nature of the infrastructure allowed us to collect data from over two hundred participants. The design also has the advantage of being generalizable to a variety of different domains simply by relabeling its inputs and outputs to represent scenarios for those domains.

4.3.1 TEST PLATFORM IMPLEMENTATION

The web interface was developed using the Google Web Toolkit [Google]. We leverage Amazon's Mechanical Turk infrastructure [Amazon] to recruit and manage participants and manage study payments by embedding our study interface in the Mechanical Turk task interface. Users found our study through the listings of *Human Intelligence Tasks* (HITs), and after accepting our HIT, they participated in the study and interacted with the system.

The user encounters several examples of system inputs and output (see Figure 4.1). He first sees the input values listed and has to click the "Execute" button so the system 'generates' the output. When he is done studying the example, he clicks the "Next Example" button to move on. Depending on the explanation condition the user is in, he may receive an explanation about the shown example.



Body Temperature: 4

Heart Rate: 2

Pace: 1

Not Exercising

Explanation
Activity predicted as *Not Exercising* because **Body Temperature** ≤ 5.0, and **Pace** ≤ 3.0

Notes
Feel free to make notes as you work with the system. It will remain from instance to instance.

Figure 4.1. Screenshot of the interface for our intelligibility testing infrastructure.

We modeled our testing infrastructure on typical sensor-based context-aware systems that make decisions based on the input values of multiple sensors. Many of these sensors produce numeric values and the applications change their behaviors based on threshold values of the sensors. For example, a physical activity recognition system could look at heart rate and walking pace. To keep our experiments and the task reasonably simple for participants we restricted the system to three input sensors that produce numeric values, we used inequality-based rules to define the output value, and constrained the output to belonging to one of two classes. In Experiment 1, for example, we defined two inequality rules that consider two inputs at a time (see Equation (4.1)). Since we did not want the lack of domain knowledge (*e.g.*, that the body temperature can rise from 36.8 to 38.3°C when weight lifting) to affect users' understanding of the system, so the inputs use an arbitrary scale of integer values: Body Temperature from 1 to 10, and Heart Rate and Pace from 1 to 5.

$$\textbf{Prediction} = \begin{cases} \text{"Excercising"} & \text{, if} \quad \text{Body Temperature} \geq 6 \quad \text{AND} \quad \text{Pace} \leq 2 \\ \text{"Excercising"} & \text{, if} \quad \text{Heart Rate} \geq 6 \quad \text{AND} \quad \text{Pace} > 3 \\ \text{"Not Excercising"} & \text{, otherwise} \end{cases} \tag{4.1}$$

Equation (4.1): Inequality-based rules for the physical activity domain.

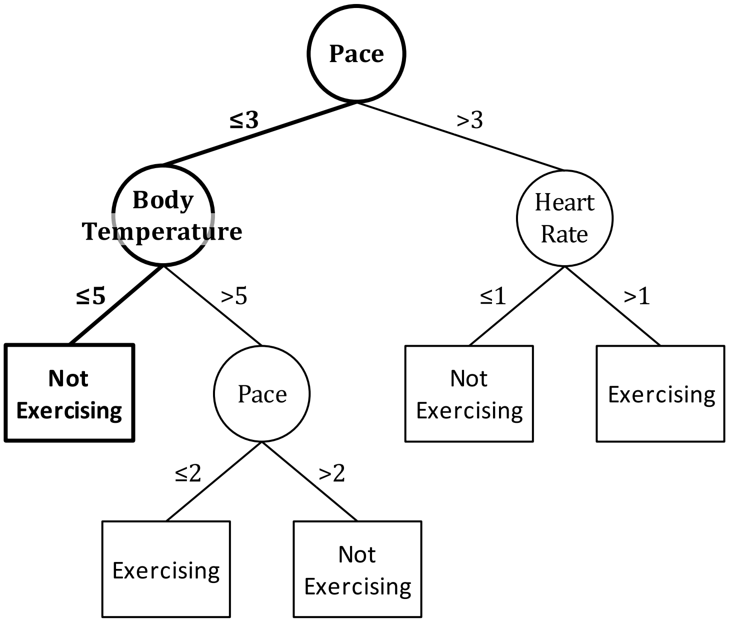


Figure 4.2. Visualization of the learned decision tree model used in Experiment 1.

As machine learning algorithms are popular in context-aware applications, our system also uses machine learning. Among the myriad of machine learning algorithms, decision trees and Naïve Bayes lend themselves to be more explainable and transparent, while others are black-box algorithms that are not readily interpretable (*e.g.*, Support Vector Machines and Neural Networks) [Nugent and Cunningham, 2005]. We chose to start our investigation using decision trees because they are easier to explain, especially to end-users who may not understand the probabilistic concepts that underlie Naïve Bayes algorithms. Using Weka's [Hall *et al.*, 2009] J48 implementation of the C4.5 Decision-Tree algorithm [Quinlan, 1993], our system learns the inequality rules from the complete dataset of inputs (250 instances from the permutations of all inputs) and outputs and models a decision tree (see Figure 4.2) that is used to determine the output value.

4.3.2 DECISION TREE EXPLANATIONS

While the decision tree is able to classify the output value given input values, we had to extend it to expose how the model is able to derive its output. The decision tree model lends itself nicely to providing explanations to the four intelligibility question types. Table 4.2 describes how the explanations were implemented.

Why: Traverse the decision tree to trace a path of decision boundaries and values that match the instance being looked at. Return a list of inequalities that satisfies the decision trace of the instance (*e.g.*, "Output classified as Not Exercising, because Body Temperature≤5 and Pace ≤3"; see Figure 2).
Why Not: Traverse the whole tree initially to store in memory all the traces that can be made. Walk the tree to find the why-trace, and find differing boundary conditions on all other traces that return the alternative output. A why-not trace would contain the boundary conditions that match the why trace and boundary conditions where it is different (*e.g.*, "Output *not* classified as Exercising, because Pace≤3, but *not* Body Temperature>5"). A full Why Not explanation will return the differences for each trace that produces the alternative output. However, so as not to overwhelm the user, we use a heuristic to return the differences of just one why-not trace, the one with the fewest differences from the why trace. Note that while this technique is suitable for small trees, it is not scalable to large trees, and heuristics should be used to look at subsets of traces.
How To: Take user specified output value, and values of any inputs that were specified. Iterate through all traces of the tree to find traces that end with the specified output value and has branches that satisfy the specified input values. If any trace is found, it identifies the satisfying boundary conditions for the unspecified inputs and returns them. Note that if there is a trace, there will only be one, since an instance can only satisfy one trace in the tree. If there are no boundary conditions for the unspecified inputs, then these inputs can take any value. If no trace is found, then there are no values for the unspecified inputs, given values of the specified inputs, to produce the desired output value.
What If: Take user's inputs and puts it through the model to classify the output. Return the output value, but since this is a simulation, do not take any action based on this output value.

Table 4.2. Algorithms for generating different types of intelligibility explanations from a decision tree model.

4.4 METHOD

Given the different factors we wanted to investigate and the flexibility of our testing infrastructure, we were able to independently test different intelligibility elements in a series of experiments. We ran Experiment 1 to explore providing different explanation types (Why, Why Not, and the control condition with no explanations). The system was presented in the context of the domain of activity recognition of exercising as described above. However, due to participants' prior knowledge of the domain, our results were difficult to interpret. So, we decided to subsequently run experiments with an abstract domain. Experiment 2 compares explanations provided to address each of the four intelligibility question types (Why, Why Not, How To, and What If) individually to investigate which are more effective in helping users gain an understanding of how our intelligent system works compared to not having explanations (None).

4.4.1 STUDY PROCEDURE

Our study consists of four sections. The first section (Learning) allows participants to interact with and learn how the system works. Two subsequent sections test the participants' understanding of the system (Fill-in-the-Blanks Test and Reasoning Test), and a final section (Survey) that asks users to explain how the system works (to evaluate the degree to which participants have learned about the system's logic) and to report their perceptions of the explanations and system in terms of understandability, trust and usefulness.

4.4.1.1 LEARNING SECTION

In the Learning section, participants are shown 24 examples with inputs and output values (see Figure 1). These examples were chosen from all possible input instances, to have an even distributed over all branches in the decision tree, and they appear in the same order to all participants. Examples were arranged in ascending order of Body Temperature, then of Heart Rate, then of Pace. Participants have to spend at least 8 seconds per example (controlled by disabling the Next Example button). Explanations are provided depending on the experimental condition. If participants receive explanations, they will receive them *automatically* when executing each example. It is important to note that explanations are only provided during the Learning section. Participants are provided with a text box to make notes in, which persist throughout the Learning section. At the end of the Learning section, users are told to spend some time studying their notes as those are not available during the rest of the study.

4.4.1.2 FILL-IN-THE-BLANKS TEST SECTION

This section tests users on their ability to accurately specify a valid set of inputs or output; they are given a single blank in one of the inputs or the output, and are given the rest of the inputs/output. There are 15 test cases, three with blank *Body Temperature*, three with blank *Heart Rate*, four with blank input *Pace*, and 5 with blank output. These test cases different from the earlier examples, and are randomly ordered, but in the same order for all participants. On seeing each test case, users have to fill in the missing input or output with a value that makes the test case correct. If an input is missing, they should provide a value that causes the given output value to be produced; if the output is missing, they provide a value that would be produced with the given input values. After providing the missing value, they are also asked to provide a reason for their response. Participants are not given any explanations during this test and, are not given the answer or told whether they are correct after they finish.

4.4.1.3 REASONING TEST SECTION

This section shows users three complete examples, and, for each example, asked to give reasons why the output was generated, and why the alternative output was not. These test case examples are different from what users have encountered before, and are randomly ordered, but are in the same order for all participants. To see if improved understanding can lead to improved trust, users are also asked how much they trusted that the output of the system is correct for each example. Participants are not given any explanations during the test and, are not given the answer after they finish.

4.4.1.4 SURVEY SECTION

The final Survey section is used to collect self-report information from users. Users provide a more detailed description of how they think the system works overall (*i.e.*, an elicitation of their mental models), and are asked 16 Likert-scale questions (see Table 4.4) to understand how users perceived about using our system, including whether they trusted and understood the system and explanations. The questions were randomly ordered to avoid order effects.

4.4.2 MEASURES

In order to see what types of intelligibility explanations would help users better understand the system, and whether this improved understanding would lead to better task performance, improved perception of the system, and improved trust in the system output, a number of measures were collected.

Task performance was measured in terms of task completion time, and the Fill-in-the-Blanks Test inputs and output answer correctness. Task completion time was measured with two metrics: total learning time in the Learning section, and average time to complete each Fill-in-the-Blanks Test question.

User understanding is measured by the correctness and detail of the reasons participants provide when they give their answers (in the Fill-in-the-Blanks Test), explain examples (in the Reasoning Test), or give an overall description of how the system works (mental model in the survey). The reasons given for each answer in the Fill-in-the-Blanks Test were coded using a rubric (see Table 4.3) to determine how much the participant understands about how the system works. Reasons are coded as Guess/Unintelligible if participants wrote they were guessing, did not write anything, or wrote something not interpretable. Reasons are graded as Some Logic if participants provided some rules or probability statement or cited past experience (*e.g.*, saying they saw something similar before) that were not inequalities with fixed numeric boundaries. This includes cases such as “Body Temperature>Heart Rate”. Reasons are coded as Inequality if participants specified an inequality of at least one of the inputs with a fixed numeric boundary (*e.g.*, Body Temperature>7). Reasons are coded as Partially Correct if participants provided only one rule with the correct input, boundary value, and relation. Reasons are coded as Fully Correct if participants get only all the sufficient rules correct, and did not list any extra ones. Each reason was coded with only a single grade (*i.e.*, the highest appropriate grade).

Understanding Code	**Description**
GUESS/UNINTELLIGIBLE	No reason given, guessed, or reason incoherent
SOME LOGIC	Some math/logic rules, probability, or citing past experience
INEQUALITY	Correct Type of rules which are inequalities of inputs with fixed numbers
PARTIALLY CORRECT	Some, but not all, of the correct rules, or extra ones
FULLY CORRECT	All correct rules, with no extra unnecessary ones

Table 4.3. Grading rubric for coding free-form reasons given by participants. Mental Models were coded using this same rubric.

There are two inequality rules (*e.g.*, Pace > 3, and Heart Rate ≥ 6) for each test case or example, so answer reasons for the Fill-in-the-Blanks Test have two components. We measure how many of these components participants learn using three coding metrics that count (i) the number of inputs the participant mentions as relevant in the reasons, (ii) the number of correct rules described, and (iii) the number of extraneous rules mentioned (0 or 1).

The reasons for the Why and Why Not questions that participants provided in the Reasoning Test were coded using a rubric similar to Table 4.3. We also recorded, on a five-point Likert-scale the participant’s level of *trust* of the correctness of the outputs for each example in the Reasoning Test.

In the survey, we asked participants to describe their overall *understanding* of how the system works. This mental model understanding is coded in a similar manner to why reasons, but not applied to specific examples.

We did a factor analysis on the 16 Likert-scale questions of system and explanation *perceptions* in the survey (see Table 4.4).

Factor	α	Likert-scale Opinions (Strongly Disagree 1 to Strongly Agree 5)
Understood System	.917	I understood the relationship between inputs and output I understood how the system works I found the system predictable I found the system easy to understand I believe I did well in the test section
Found System Confusing (Negated)	.722	I found the system confusing I found the system complicated I found the system hard to remember
Liked System / Found it useful	.648	I learned something new from interacting with this system I liked interacting with the system
Explanations Difficult (Negated)	.529	I found the explanations insufficient I found the explanations confusing I found the explanations too detailed
Explanations Useful	.816	I found the explanations appropriate I found the explanations useful
Understood Explanations	N.A.	I understood the explanations

Table 4.4. Likert-scale questions of perception grouped into six factors with Cronbach's α reliability computed. The former three factors are regarding the system, and the latter three factors only apply to participants who viewed Intelligible versions of the system.

4.5 EXPERIMENT 1

Our first experiment focused on providing answers to hypotheses H1 and H2; whether Why explanations would lead to improved user understanding, trust, perception, and performance more than having no explanations, and H2 regarding providing Why Not explanations being better than no explanations, but not as good as Why explanations. We chose the domain of activity recognition of exercise, of which users would have a reasonable understanding. Mapping to the generalized abstract system described earlier, the system takes on the role of a wearable device that can measure the wearer's Body Temperature, Heart Rate, and walking or running Pace, and classify whether the wearer is exercising (Equation (4.1)). The first rule can be satisfied during strength training (*e.g.*, weight lifting) that does not require much walking about, but can raise body temperature, while the second rule can be satisfied by running.

Participants in the no explanation (None) condition did not receive any explanations, and could only execute each example and move on. Participants in the Why condition receive Why explanations automatically along with the output value when they execute each example by clicking the "Execute" button. Participants in the Why Not condition receive a Why Not explanation in place of a Why explanation.

4.5.1 PARTICIPANTS

53 participants were recruited, aged from 18 to 57 (M=29.8). There were 18 participants in the None condition, 18 in the Why condition, and 17 in the Why Not condition. We removed from the analysis any responses of participants who took fewer than 15 minutes (one participant in the None condition) or longer than 50 minutes to complete the four sections. This was done to filter out participants who just click through the steps without thinking, and to leave out participants who may be distracted while performing the task and take too long. On average, participants took 34 minutes to complete the study. Participants were each given $3 for completing the study ($1 base and a $2 bonus to motivate performance). A further $2 was offered to a few participants who participated in interviews conducted soon (up to a few days) after completing the task.

4.5.2 RESULTS

To analyze participants' ability to apply their understanding, the number of correct answers per participant was summed and a Tukey HSD pair-wise test was performed. The number of correct answers was the dependent measure. The analysis showed significant differences in accuracy between explanation types (F[2,84]=8.85, p<.001; see Figure 4.3). To analyze participants' ability to formalize their understanding, their reasons were coded using the coding scheme in Table 4.3 and dummy variables were generated indicating: Inequality or better (0 or 1), Partially or Fully Correct (0 or 1), and Fully Correct (0 or 1). The analyses were done with the reason coding as the dependent measure and with condition as a fixed effect. Participants were modeled as a random effect and nested within condition. A Tukey HSD pairwise test of the occurrences of each coded score shows that providing explanations leads to more correct answers than not providing any (contrast of None with Why and Why Not: F[1,50]=15.1, p<.001). However, there was no significant difference in the number of correct answers between Why and Why Not explanation types.

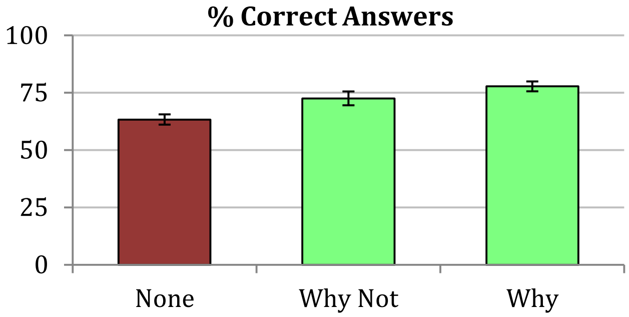


Figure 4.3. Participants receiving explanations (in the Learning section) answered significantly more questions correctly in the Fill-in-the-Blanks section.

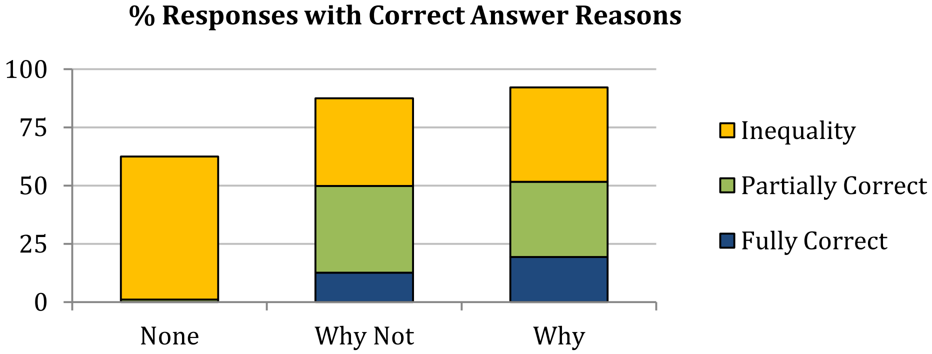


Figure 4.4. Percent of reasons coded as *Inequality*, *Partially Correct*, or *Fully Correct* in the Reasoning Test section.

Using the grading coding scheme in Table 4.3 on the Why reasons provided in the Fill-in-the-Blanks Test, we found that participants in the Why and Why Not conditions were able to produce more Partially Correct reasons compared to those in the None condition (F[1,50]=27.4, p<.001) (see Figure 4.4). Participants in the Why condition produced more Fully Correct reasons compared to None and Why Not (F[1,50]=10.8, p<.002). There were no significant differences between Why and Why Not. A similar pattern was found in the Reasoning Test section Participants in the Why condition had a higher level of trust than those in None (F[1,49]=8.98, p<.005), while those in the Why Not condition did not. The survey measures on overall mental model or perceptions of the system and explanations did not reveal significant differences.

4.5.3 DISCUSSION AND IMPLICATIONS

The generally poor trust in the system could be due to occasional examples that follow the system rules, but may not be 'natural' (*e.g.*, high Body Temperature and low pace predicted as "Not

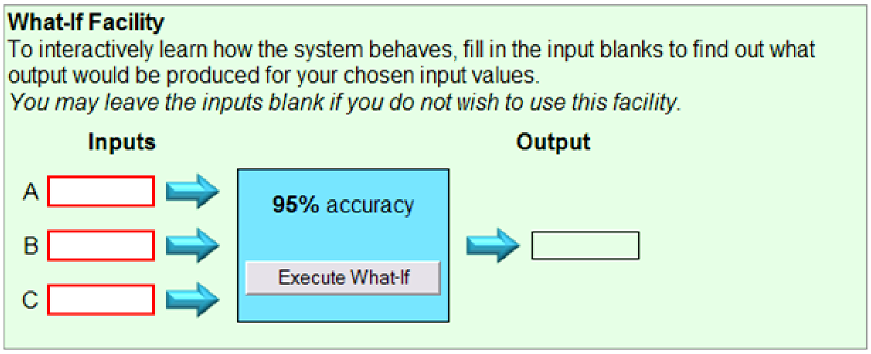
Exercising"). The answer and reason results indicated that providing explanations lead to better understanding and trust of the system with less disagreement about the system output. However, in their provided why reasons, several participants alluded to the domain of physical activity and physiology to explain how the inputs (Body Temperature, Heart Rate, and Pace) should relate to whether the device wearer was "Exercising" (*e.g.*, *"moving & high [body temperature], looks like running so I upped the [heart rate]"*). Furthermore, most responses specified the inputs as "high" or "low" rather than specifying numeric boundaries (*e.g.*, *"heart rate is low, so must be a high pace along with high body temperature to predict exercising"*). This suggests that having prior knowledge would lessen participants' effort to be precise about their understanding. To mitigate the effects of prior knowledge, and to support more generalizability to other domains, we decided to anonymize the inputs and outputs with an abstract system.

4.6 EXPERIMENT 2

Our second experiment focused on comparing the effectiveness of different explanations types for each of the 4 intelligibility questions. Using the explanation algorithms described in Table 1, we can isolate these explanations for each condition.

4.6.1 METHOD

This experiment followed the procedure of Experiment 1. For the None, Why, and Why Not conditions, participants see the same interface as in Experiment 1, but with the inputs obfuscated as *A*, *B*, and *C*, and the output values relabeled to *a* and *b*.

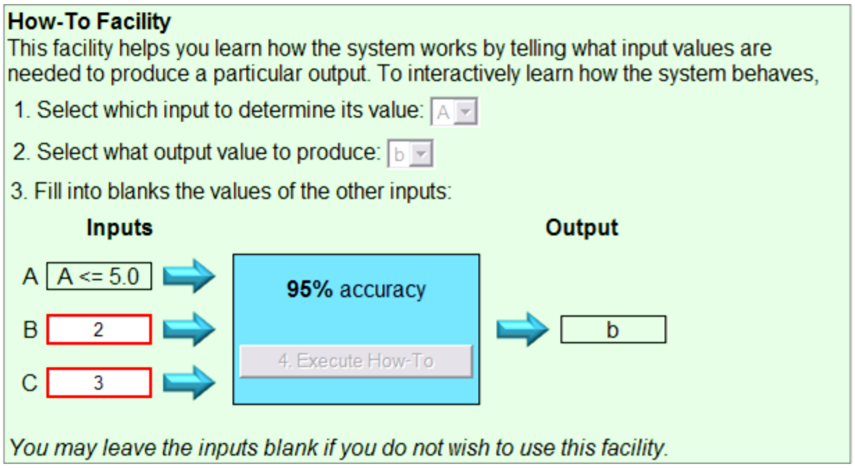


What-If Facility

To interactively learn how the system behaves, fill in the input blanks to find out what output would be produced for your chosen input values.

You may leave the inputs blank if you do not wish to use this facility.

Figure 4.5. What If explanation facility. Participants would get to freely enter values for the inputs A, B, and C, and get the system to simulate what the output would be.



How-To Facility

This facility helps you learn how the system works by telling what input values are needed to produce a particular output. To interactively learn how the system behaves,

1. Select which input to determine its value: A
2. Select what output value to produce: b
3. Fill into blanks the values of the other inputs:

A: A <= 5.0
B: 2
C: 3

Output: b

You may leave the inputs blank if you do not wish to use this facility.

Figure 4.6. Participants in the How To condition view this facility. By specifying two of the input values and an output value, they can inquire the system to indicate possible values of the remaining input.

Participants in the What If condition receive a What If interaction facility (see Figure 4.5) instead of an explanation to let them see the output given their choice of inputs. Participants in the How To condition received an interactive facility (see Figure 4.6) to determine how to get the system to produce a chosen output value. To control for the number of examples encountered, participants in the What If and How To conditions only get 12 complete examples (the even-numbered examples of other conditions), and can invoke their respective intelligibility facilities 12 times to see a total of 24 examples (similar to the other conditions). For each condition, the explanations or explanation facilities will always appear as each example is executed.

4.6.2 PARTICIPANTS

158 participants were recruited, aged from 18 to 72 (M=31.9). There were 26-37 participants in each of the 5 conditions: None (31); Why (30); Why Not (31); How To (29); What If (37). On average, participants took 33 minutes to complete the study (similar to Experiment 1, they were required to complete the study within 15 to 50 minutes). Compensation was identical to Experiment 1.

4.6.3 RESULTS

We analyzed the results by using the Tukey HSD pairwise test, looking for differences between groups for our previously described metrics. Compared to participants in the None, What If and How To conditions, participants in the Why and Why Not conditions had *more correct* answers in the Fill-in-the-Blanks tests, provided *better reasons*, and reported having a *better understanding* of the system. Participants in the Why and Why Not conditions had an accuracy of 80.0% and 74.2%, respectively, compared to 61.7% for the None condition (F[1,152]=51.6, p<.001; see Figure 4.7). More of their answer reasons were coded as at least Inequality type rules (Inequality: F[1,153]=198, p<.001), Partially Correct (F[1,153]=195, p<.001) and Fully Correct (F[1,153]=108, p<.001). Finally, the self-reports of understanding for Why and Why Not were 3.14 and 2.79, respectively (see Figure 4.10a).

Participants in the Why condition further distinguished themselves from Why Not by giving more Fully Correct reasons (contrast of Why with Why Not: F[1,153]=23.2, p<.001), and trusting the system output more (contrast of Why with None: F[1,153]=8.26, p<.001 vs. contrast of Why Not with None: p=n.s.) with means of 3.26, 3.0 and 2.46 for Why, Why Not and None, respectively (see Figure 4.10b). However, these participants also took the longest to answer each Fill-in-the-Blanks test case (M=26.3 seconds, compared to M=22.0 and M=17.0 for Why Not and None, respectively) (contrast of Why with None: F[1,145]=9.32, p<.003 *vs*. contrast of Why Not with None: p=n.s.).

Surprisingly, participants in the Why Not condition were *not* significantly better at providing Why Not reasons than Why reasons. While participants in the What If condition were indistinguishable from those in the None condition across all of our metrics, we did find that participants in the How To condition were able to understand the types of rules used in the system better than participants in the None condition (answer reasons coded as Inequality or better: F[1,153]=15.6, p<.001).

To identify why participants in the Why Not condition understood less about the rules than Why, we coded the quality of answer reasons on the number of inputs and rules mentioned. Participants in the Why condition provided more correct rules (M=1.19 *vs.* M=0.79; F[1,59]=6.16, p<.02) while those in the Why condition provided fewer extraneous rules (M=0.11 *vs.* M=0.23; F[1,59]=8.276, p<.006).

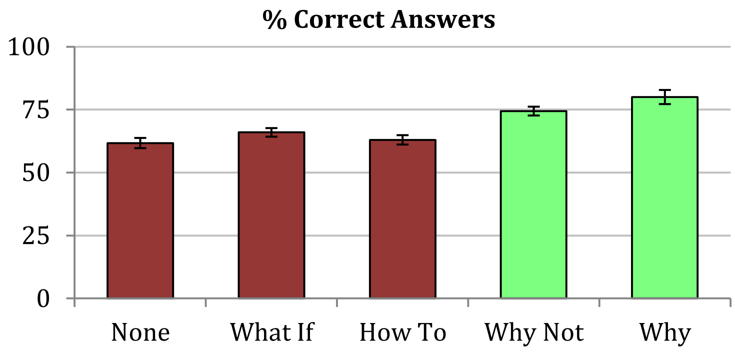


Figure 4.7. Percent of correct answers in the Fill-in-the-Blanks test section, by condition. Different colors indicate statistically significant differences.

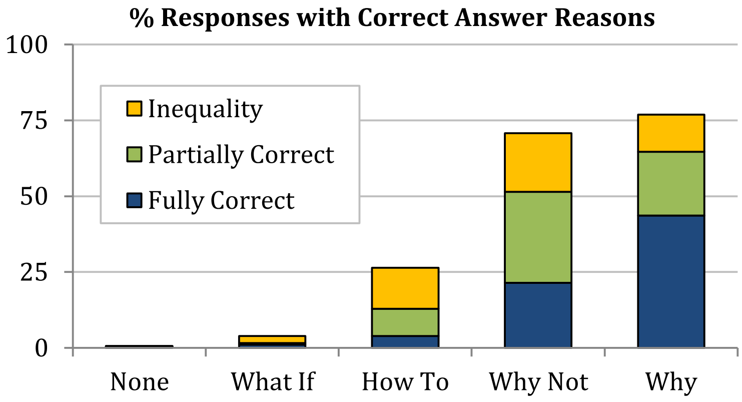


Figure 4.8. Percent of reasons coded as *Inequality*, *Partially Correct*, or *Fully Correct* in the Fill-in-the-Blanks Test section for each condition.

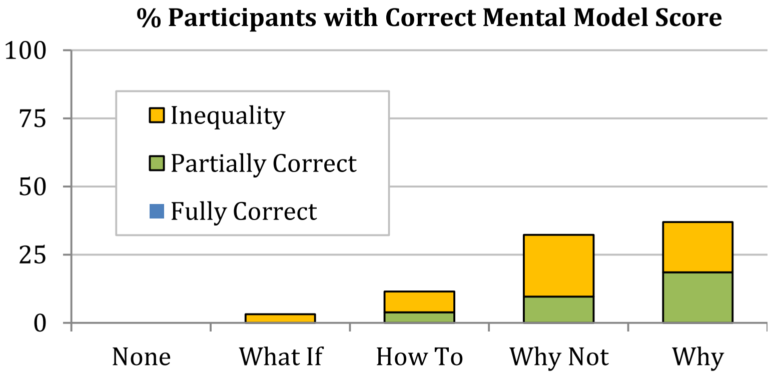


Figure 4.9. Overall understanding of the system was similar to the understanding in-situ of individual examples, but responses were less precise (fewer correct descriptions).

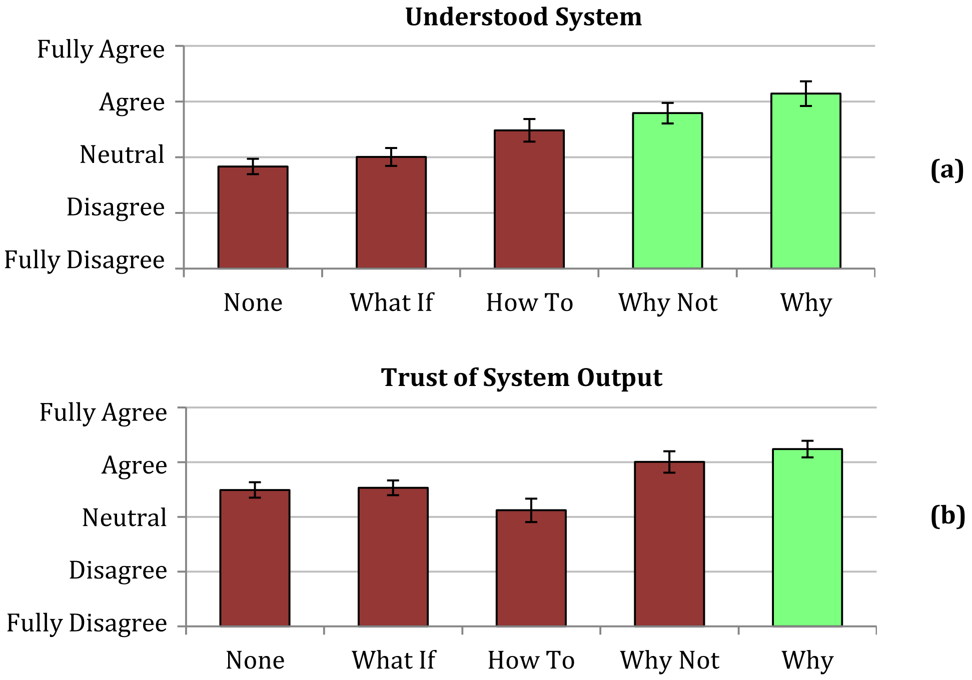


Figure 4.10. Self-reports of (a) understanding and (b) trust, by condition. Different colors indicate significant differences.

4.6.4 DISCUSSION AND IMPLICATIONS

The results in Experiment 2 validate those in Experiment 1 with a more generalized abstract domain, while not suffering from confounds due to prior domain knowledge. The Why and Why Not explanations improved participants' understanding, increased their trust in the system, and their task performance. Examining the user reasons, we found that automatically generated Why explanations allowed users to more precisely understand how the system functions for individual instances compared to Why Not explanations. This is in spite of the Why Not explanations being logically equivalent to Why explanations since flipping the *not*'s in the former can derive the latter. Moreover, we found that the Why Not participants tended to provide fewer correct rules (more participants could only provide one correct rule instead of two) for the answer reason, or provide extraneous inputs and rules that the system did not consider for the respective test cases, as compared to the Why participants. These indicate that Why Not participants tended to learn only part of the reasoning trace, and did not associate the two rules together, but treated them separately. This failure in rule conjunction could be due to the inclusion of negative wording (*i.e.* "but" and "not") in the Why Not explanation. The mental effort to understand the Why Not explanation and create such a rule conjunction is certainly more than those in the Why condition had to expend, which could explain the differences we observed.

Neither the How To nor What If explanations showed much benefit over not having explanations. Some participants expressed their difficulty in using these explanation types, *e.g.*, "*I really don't think I used it cause I did not understand it*"; "*The first few [times, I did] not even realize what the facility was for*." Participants receiving What If explanations did not optimize their selection of examples, with some users even selecting input values out of range (*e.g.*, A=100). Given the abstract and mathematical nature of the experimental setup, without any reasoning trace (unlike Why, Why Not, How To), almost none of these participants proposed inequality rules as reasons, similar to those in the None condition. However, as with the effect of domain knowledge (in Experiment 1), participants who did not receive reasoning traces did consider the inequality rules, but just not correctly (see Figure 4.5).

Our results suggest that developers should provide Why explanations as the primary form of explanation and Why Not as a secondary form, if provided. Our results may suggest the ineffectiveness of How To and What If explanations, but these explanation types may be more useful for other types of tasks, particularly those relating to figuring out how to *execute* certain system functionality, rather than interpreting or *evaluating*.

4.7 GENERAL DISCUSSION

We now discuss the findings of our two experiments and their implications for real world context-aware systems.

4.7.1 IMPACT OF PRIOR KNOWLEDGE

We found in Experiment 1 that participants formed less accurate and precise mental models of the system, compared to those in Experiment 2. This could be due to participants applying their prior knowledge of exercising to understanding how the system works and not paying careful attention to the explanations, as evidenced by the reasons they provided. This persistence of mental model was also shown in [Tullio *et al*., 2007] where participants received explanations, over time, of how an interruptibility system worked. As many real context-aware applications are based on common everyday activities, users may have strong prior knowledge of the domains although weak understanding of the applications, and may also not diligently learn from the provided explanations. One way to address this could be to learn from the knowledge-based systems community, and provide deeper *justification* [Gregor and Benbasat, 1999] explanations to help users understand why the system behavior may be different from typical everyday understanding.

4.7.2 FROM THE LAB TO THE REAL WORLD

Our intelligibility test infrastructure differs from real applications in that users would have different goals when asking either of the intelligibility question types. In reality, users would ask Why questions when they lack an understanding of how the application works, but Why Not questions when they expect certain results that the application did not produce. This distinction in user expectations and goals was not present in our lab study. Therefore, even if Why Not explanations are found to be less effective than Why explanations, for real systems, users may prefer the former explanation type to bridge gaps in their understanding and improve their trust and acceptance of the system.

In order to investigate how our findings play in a real-world setting, we have developed an intelligible, context-aware plugin [Lim and Dey, 2012a] for the AOL Instant Messenger (AIM) that uses predictions of buddy responsiveness to instant messages (based on [Avrahami and Hudson, 2006]). In a future longitudinal deployment we plan to investigate how explanations affect usability and acceptability.

4.7.3 IMPLICATIONS FOR CONTEXT-AWARE APPLICATIONS

While our intelligibility test infrastructure has some characteristics of context-aware systems, real context-aware applications are more complex and several issues would have to be handled regarding the provision of explanation types. Firstly, applications that use decision tree models tend to have much larger trees learned from possibly hundreds of features, and it would not be scalable to generate explanations from them. For example, a tree of depth 13 could lead to the Why traces that have over 10 inequality relations. The explanations returned would be too long for users to assimilate and remember. One way to deal with the larger tree size is to just provide subsets of reasons in the explanations. For example, the Why trace could just provide the top 5 inequality relations ranked by how much each relation affects the prediction accuracy. Providing subsets of explanations would provide users with only partial understanding of each application behavior instance, and users may have to interact with the system longer before understanding the system better. One way to reduce overall learning time may be to start new users with higher-detail explanations, then progress to less detail the more they interact with the system.

While our setup dealt with decision tree learners, the naïve Bayes classifier is another popular learner used in context-aware applications. Even though they are not as intuitive as decision trees, Naïve Bayes models can be interpretable, and there are several visualizations to explain them (*e.g.*, nomograms [Mozina *et al*., 2004]). However, some learners (*e.g.*, Support Vector Machines with Gaussian kernels, Neural Networks) are considered black-boxes [Nugent and Cunningham, 2005] and are not inherently interpretable. Fortunately, there have been some attempts to make them explainable using decision trees or rules (*e.g.*, [Andrews, Diederich, and Tickle, 1995]). We can then use the same techniques to provide explanations for systems based on decision tree models.

Another issue with real systems is that users may not like to receive explanations *all the time*, but *on demand* instead, because the former may be too obtrusive. In Chapter 9, we performed a study to compare if users can still benefit sufficiently from explanations if they get to choose when and how often they can receive explanations, and if this usage of explanations can lead to improved learning.

Our results suggest the effectiveness and importance of providing Why and Why Not explanations over How To and What If. The former two deal with Norman's gulf of evaluation, while the latter two deal with the gulf of execution [Norman, 1988]. While we feel that this dichotomy should remain true for informative context-aware systems (*e.g.*, applications to determine interruptibility of others to inform onlookers [Avrahami and Hudson, 2006; Tullio *et al*., 2007]), systems that are more pro-active (*e.g.*, applications that send notifications based on the user's interruptibility) may benefit more with the How To and What If explanations. With those explanations, users would be better informed of how they can carry out their tasks.

4.8 CONCLUSIONS AND FURTHER WORK

We have described a large controlled study comparing the provision of explanations addressing four explanation type questions (Why, Why Not, How To, and What If). We developed a web-based platform that provides a functional input-output interface of an intelligent system prototype that provides different types of explanations. Our findings suggest that providing reasoning trace explanations for context-aware applications to novice users, and in particular Why explanations, can improve user's understanding and trust in the system.

Our results of the relative strengths and weaknesses of each explanation type came from a *between-subjects* study, but to gain an insight into which explanation type individual users may prefer, we wish to run a *within-subjects* study, where each participant sees multiple explanation types. In Chapters 7 and 9, we investigate this with an intelligible context-aware mobile application, which provides several explanation types.

Furthermore, though our results do not show the effectiveness of How To and What If explanations, we believe they may be more useful given better motivating scenarios and better interface design. Therefore, we continued to pursue our investigations into these explanation types in later work (Chapters 5, 6, 7, and 9), and specifically sought out a user friendly interface for explanations in Chapter 7.

We next sought to widen the scope of intelligibility to include more questions that users may ask of context-aware applications. In Chapter 5, we expand on four intelligibility question types to include 11 question types for our taxonomy of Intelligibility.

5 Assessing Demand for Intelligibility

This chapter is an extension of the work presented in:

> Lim, B. Y. and Dey, A. K. (2009). Assessing Demand for Intelligibility in Context-Aware Applications. In *Proceedings of the 11th international Conference on Ubiquitous Computing (Orlando, Florida, USA, September 30 - October 03, 2009). Ubicomp '09*. ACM, New York, NY, 195-204.

Abstract. Intelligibility can help expose the inner workings and inputs of context-aware applications that tend to be opaque to users due to their implicit sensing and actions. However, users may not be interested in all the information that the applications can produce. Using scenarios of four real-world applications that span the design space of context-aware computing, we conducted two experiments to discover what information users are interested in. In the first experiment, we elicit types of information demands that users have and under what moderating circumstances they have them. In the second experiment, we verify the findings by soliciting users about which types they would want to know and establish whether receiving such information would satisfy them. We discuss why users demand certain types of information, and provide design implications on how to provide different explanation types to make context-aware applications intelligible and acceptable to users.

5.1 Introduction

In Chapter 4, we found that some types of explanation were more effective than others in improving users’ understanding and trust of a context-aware intelligent system. However, it was not clear what information users actually want to know and will ask about, and whether there are more explanation types than we had previously considered. In this work, we explored and assessed a *taxonomy* of user demand for intelligibility: which types of questions users want answered, and how answering them improves user satisfaction of context-aware applications. User satisfaction is obviously crucial for adoption and acceptance of such technologies.

To make context-aware applications intelligible so that they can expose their inner functions to the end-user, much research has looked into how to generate explanations from the underlying application models and deliver them to users (*e.g.*, [Cheverst *et al*., 2007; Ko and Myers, 2009; Kulesza *et al*., 2009; Lim and Dey, 2009]). However, little work has been done to compare the impact of different types of explanations or in the domain of context-aware computing. Users may not be receptive to these explanations, especially when they end up using the applications in ways for which they were not designed [Orlikowski, 2000], and when those explanations do not adapt to varying situations of use. Thus it is important to explore information demand from the user's perspective lest effort is wasted in implementing explanations that would see little use.

Researchers have explored what users want to know in other domains. McGuinness and colleagues [Glass, McGuinness, and Wolverton, 2008; McGuinness *et al*., 2007] have identified information need factors that influence the level of trust in adaptive agents. They used interviews to identify explanation requirements and rank question types according to their helpfulness. Gregor and Benbasat's [1999] meta-review investigates explanation types that users of knowledge-based systems (KBS) would like to have. While adaptive agents and KBS are similar to context-aware applications (which may also use agents or knowledge bases and rules), they are work-oriented, while context-aware applications are targeted for everyday use, for many more situations and a wider range of users, and under more situations [Abowd, Mynatt, and Rodden, 2002]. Thus we need to explore how these different requirements would lead to different intelligibility needs.

The chapter is organized as follows: we discuss how supporting intelligibility by providing explanations that users want, has the potential to increase user satisfaction and thus acceptance of context-aware applications. We then describe our experimental design that uses surveys and scenarios to expose users to a range of experiences with context-aware applications. We present two experiments that investigate what types of information users want. In the first experiment, we elicit the types of information users are interested in and under what moderating circumstances. In the second experiment, we validate our findings by presenting users with 11 information types as intelligibility features in a controlled study and measure their impact on user satisfaction. We end with a discussion of why users of context-aware systems demand certain types of information in different situation, and provide design recommendations for providing different information types to make context-aware systems intelligible and acceptable to users.

5.2 HYPOTHESES AND APPROACH

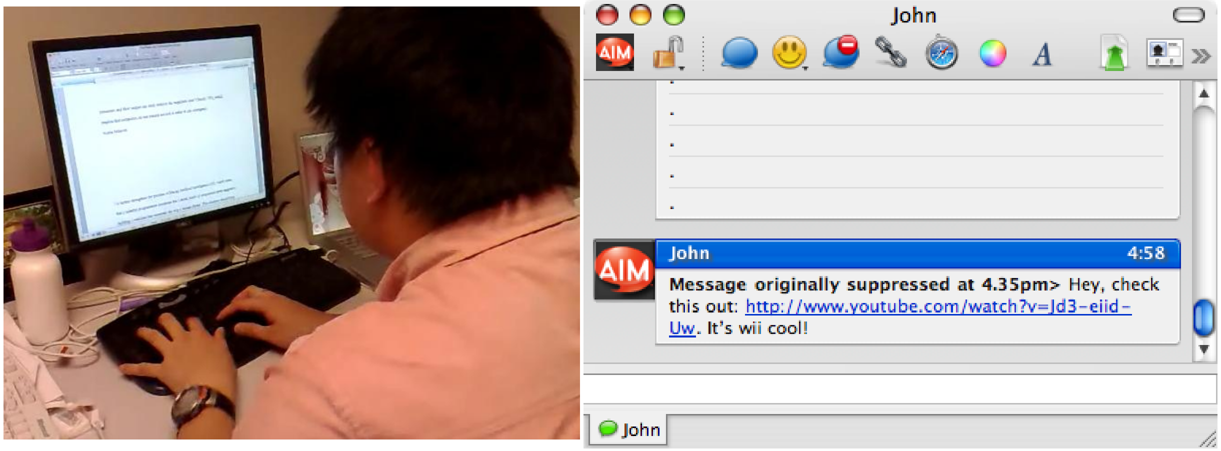
We hypothesize that there are different types of information in which users are interested, for different context-aware applications, and different situations. Since people ask information seeking questions due to cognitive disequilibrium [Graesser and McMahen, 1993] and to correct knowledge deficits [Van der Meik, 1987], we believe that satisfying these information demands through intelligibility can lead to better satisfaction when using these applications and improved adoption and acceptance. In order to elicit the information demands users have for context-aware applications under various situations, we conducted a study of the demand for explanations and different types of information in several scenarios users may find themselves in as they use context-aware applications.

Using described scenarios instead of actual field deployments allows us to quickly and more effectively study and understand the impact of different information on intelligibility and satisfaction, without having to implement and deploy a variety of applications, any of which could fail for reasons independent of our main focus. Next we describe four applications we use to focus our scenarios. For each application, the scenarios intentionally span a range of incorrect, appropriate and unexpected or anomalous, but not necessarily wrong behavior, to probe directly at the issues of intelligibility and satisfaction.

5.3 SETUP: SCENARIOS OF FOUR CONTEXT-AWARE APPLICATIONS

To investigate the demand for intelligibility in the space of context-aware applications, we selected four prototypical context-aware applications: (i) a desktop interruption management application (an Instant Messenger plugin), (ii) a remote person monitoring peripheral display (Digital Family Portrait), (iii) a context-aware reminder application (CybreMinder), and (iv) a mobile context-aware tour guide (CyberGuide). All applications in this study behave according to models of learned decision trees.

5.3.1 INTERRUPTION MANAGEMENT



John 4:58

Message originally suppressed at 4.35pm> Hey, check this out: http://www.youtube.com/watch?v=Jd3-eiid-Uw. It's wii cool!

Figure 5.1: (Left) Screen capture of a five-second video clip for the IM Auto-Notification application survey, showing the user rushing to meet a deadline. (Right) Screenshot of a non-work IM message which had been suppressed and delivered later.

We designed the instant messenger (IM) auto-notification plugin based on recent work on a predictive model to determine how long a buddy would take to respond to a message [Avrahami and Hudson, 2006]. Our application uses the responsiveness prediction to determine the subject's interruptibility [Fogarty *et al.*, 2005], and either *forwards* or *suppresses* incoming IM messages. We developed four main scenarios for this application where the subject is in various states of availability:

1. Rushing to reach an imminent deadline,
2. Taking a break and surfing the Internet,
3. Reading a work-related book, and
4. Returning from a protracted informal meeting.

For each scenario, the user receives an IM message from

- A colleague regarding critical work, or
- A friend regarding a fun video.

There are 16 scenarios (4 availability × 2 received messages × 2 application actions).

5.3.2 Remote Monitoring

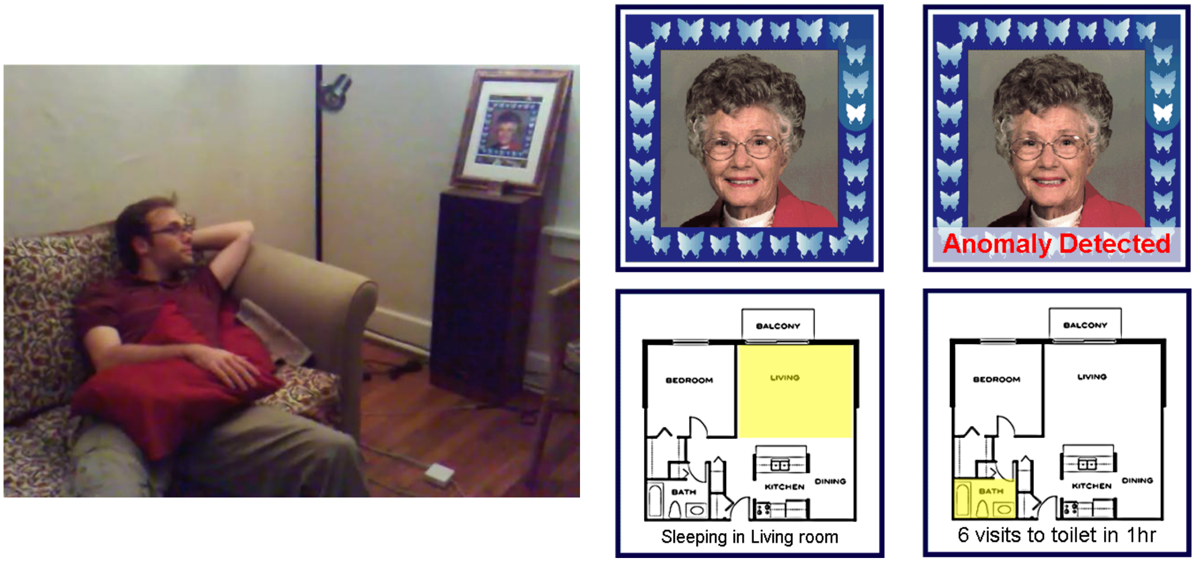


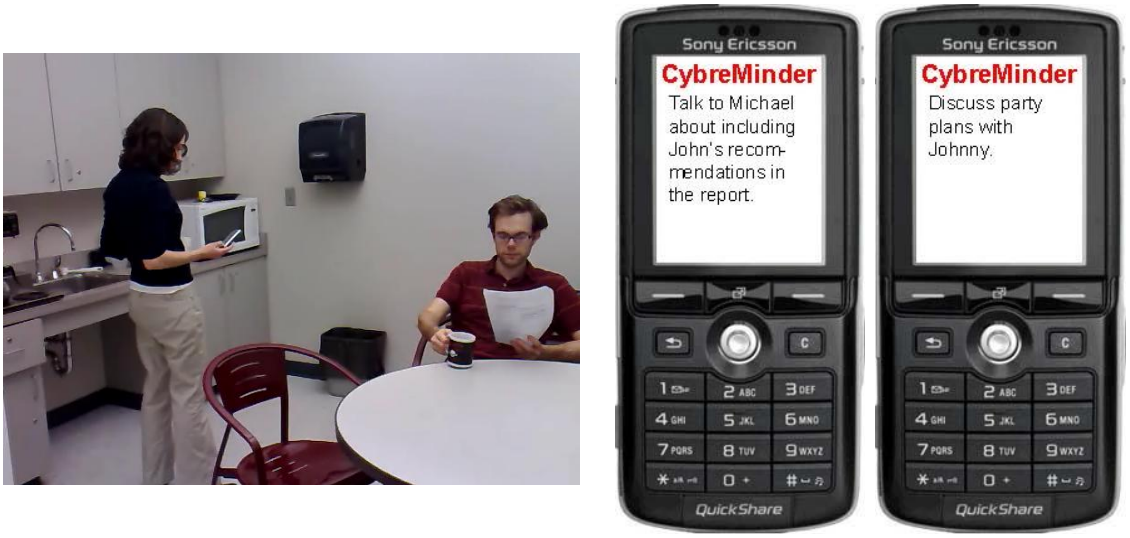
Figure 5.2: (Left) Screen capture of a five-second video clip for the Elderly Remote Monitoring application survey, showing the user casually glancing at the display. Screenshots of a normal event (Middle) and an anomalous event (Right).

We used the Digital Family Portrait [Mynatt *et al.*, 2001] as an example for remote monitoring systems. It leverages a picture frame to present the current status of an elderly family member as he or she goes through daily life living independently in her home, to remote loved ones. Our rendition of the Digital Family Portrait is based on a decision tree model which we define as several small subtrees, each addressing groups of scenarios. We present a subset of what the sensors on the elder's body and in the home are described as detecting:

1. Whether the family member has fallen,
   Whether there is a fire;
2. How many times the toilet has been used recently,
   Whether the usage frequency is anomalous,
   Whether the system thinks this could be a symptom of incontinence;
3. Whether the family member is watching TV,
   Whether the family member is sleeping
4. Whether the family member's house is vacant,
   Whether there is an intruder.

For this application, there are a total of 13 scenarios.

5.3.3 REMINDER



Talk to Michael about including John's recom-mendations in the report.

Discuss party plans with Johnny.

Figure 5.3: (Left) Screen capture of a five-second video clip for the Reminder application survey, showing the phone triggering at the pantry. Screenshots of a work-related reminder (Middle) and personal reminder (Right).

We used CybreMinder [Dey and Abowd, 2000] as an example for reminder systems. CybreMinder is a context-aware reminder application that considers combinations of contexts, such as location, time, and collocation, to trigger reminders. It is based on several personal and environmental sensors, and triggers reminders based on the satisfaction of one of several rules (modeled as a decision tree). We developed scenarios that would relate to three types of reminders (mentioned in [Dey and Abowd, 2000]):

1. Reminder to discuss an important issue when the user and a colleague serendipitously meet (collocation trigger);
2. Reminder to take the umbrella when it is forecasted to rain and the user is approaching the front door (location and information trigger); and,
3. Reminder to discuss party planning with a friend when the user and the friend are free, and the user is at the office (complex trigger).

We developed 13 scenarios based on these three reminders.

5.3.4 TOUR GUIDE RECOMMENDER

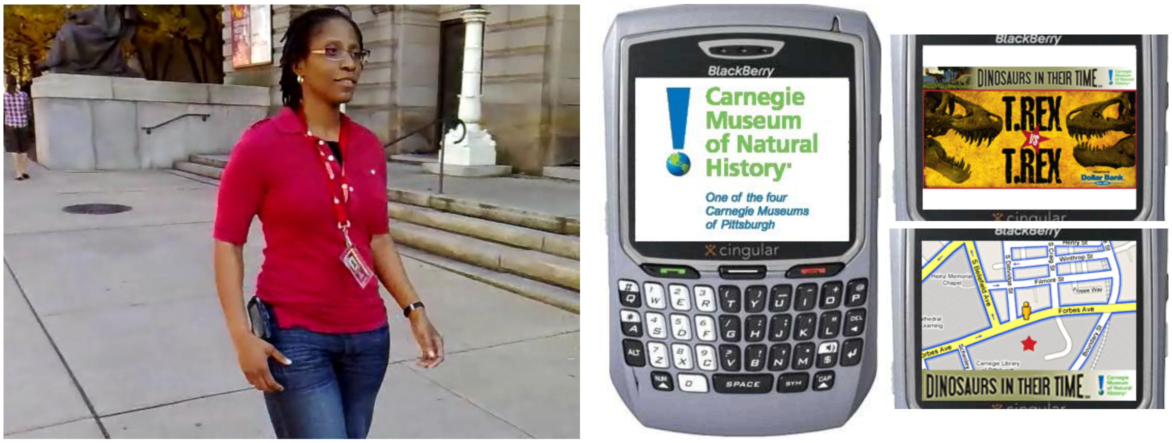

BlackBerry
Carnegie Museum of Natural History
One of the four Carnegie Museums of Pittsburgh
cingular
DINOSAURS IN THEIR TIME
T.REX vs T.REX
Forbes Ave
Dollar Bank

Figure 5.4: (Left) Screen capture of a five-second video clip for the Tour Guide Recommender application survey, showing the user walking by the museum, a point of interest. (Right) Animated screenshots recommending a dinosaur exhibit at the museum.

We used CyberGuide [Abowd *et al*., 1997] as an example for tour guide systems. CyberGuide is a mobile context-aware tour guide that uses context to recommend attractions to a user. Our rendition of CyberGuide considers the contexts of location (where the user is currently located), keywords (that the user has recently used), and navigation information like traffic. We use three settings for a user with a keen interest in museums and dinosaurs visiting an unfamiliar city:

1. Walking by museum with a dinosaur exhibit;
2. Having a conversation with a friend talking about museums and dinosaurs; and,
3. Meeting a friend at his home with the application recommending a route to the destination.

We developed 12 scenarios based on these three settings

5.4 EXPERIMENT 1: ASSESSING DEMAND FOR INFORMATION TYPES

Based on these four applications and the scenarios we developed, we conducted our first experiment on the intelligibility of context-aware applications to investigate what questions users want to ask in the various scenarios.

5.4.1 METHOD

We created one survey for each of the four prototypical context-aware applications. At the beginning of each survey, the respective context-aware application is described to the participant. Its functionalities are described, but not explained or elaborated on (*e.g.,* participants are not told where it gets its information from). Depending on the application survey, participants are shown 3-4 scenarios, described from a first-person perspective that places participants in certain circumstances (*e.g.*, rushing to complete a report due in half an hour for the IM application). The scenarios are represented by short 5-second video clips (*e.g.*, see Figure 5.1 left) and short textual descriptions about what is happening. After a scenario is presented, participants are shown 2-5 (depending on the application and scenario) instances of the scenario, one at a time, with different application responses, represented as screenshots along with text (*e.g.*, see Figure 5.1 right), where the behaviors may be appropriate, strange, or incorrect. For each application response, participants are reminded about the scenario and can replay the video, if necessary.

To mitigate order effects, the order of scenarios was randomized for each participant. For each scenario, we posed several questions (see Table 1) to ascertain what the participant thought and felt about the application response, and what information they would want to know, if any, for the situation. Except for a question on application satisfaction with a 7-point Likert scale response, the rest of the questions were free text response.

Measure	Survey Question	Response
Application Satisfaction	I am satisfied with the application response	7-point Likert scale
Action	What will you do?	Free text
User Feeling	How do you feel about what the application did?	
Information Demand	What information or knowledge would you like to know about what the application did or why it behaved this way?	

Table 5.1: Questions posed to participants for each application response scenario to find out what they think the application response, what they think is happening, and their information demand for each scenario.

We recruited 250 participants (47% female; ages 18 to 61, M=29.5) from Amazon Mechanical Turk, and paid them $4 for completing a survey. Participants were divided among the four applications surveys, and survey analysis was conducted between-subject.

5.4.2 CODING ANALYSIS: EXPLANATION TYPES AND MODERATORS

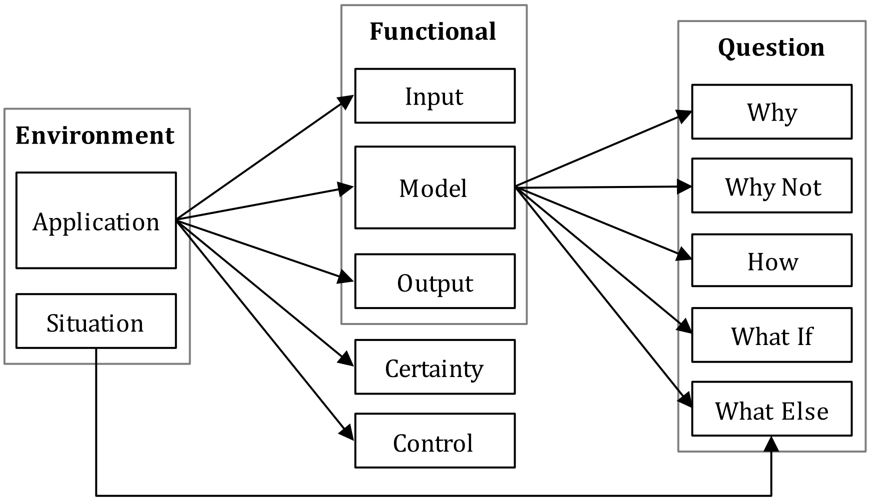


Figure 5.5: Hierarchical representation of explanation types that users want to know.

We coded the participant responses to the free response questions to determine whether participants wanted more information regarding the application or situation, and what type of information they wanted. Using the open coding method of grounded theory [Strauss and Corbin, 1990], and drawing from question types employed previously in [Antifakos *et al.*, 2005] and Chapter 4, we derived a set of explanation types that users are interested in for context-aware applications. In the rest of the chapter, we shall refer to these types as *explanation types*. Figure 5.5 presents them in terms of a hierarchy and Table 5.2 shows the coding scheme used.

Theme	κ	Values / Description		Example Participant Text
Impression	.91	–	Negative (Useless / Dissatisfied / Irritated / Frustrated)	"Angry" "Bad" "I'm pissed off it could have cost me my job" "I feel cheated" "I'm disappointed in the application"
		0	Neutral / equivocal	"It's ok." "fine"
		+	Positive (Useful / Satisfied)	"Good. Positive." "I love it, works like a charm."
Trust of Device	.91	0	Abandon it, Complain, Lost Trust, Doubt it, Dissatisfied / believe it is wrong	"I wouldn't be paying attention to it anymore" "Something must be wrong with one of the sensors." "It may have misinterpreted an action."
		1	Satisfied / Believe / Trust	"The application performed properly." "I'm satisfied." "It's okay, it was my fault."

Theme	κ	Values / Description		Example Participant Text
Information Demand	.84	0	None / Not necessary	-
		1	Too much info already / Overwhelmed	"It is more than I would want to know."
		2	Yes / Not enough info/details	*(See below)*
Domain of Explanation	.95	0	None	-
		1	Application, Device, Sensors (includes logic)	*(See below)*
		2	Situational / Event	"My status and Johnny's priority." "How often the person moved, or what the rate of respiration was." "Our locations and distance."

Table 5.2: Coding scheme for Experiment 1. The first two themes indicate participants' thoughts on the scenario, and the remaining themes indicate their information needs. Most of the participant text responses were coded by one coder, with a 10% random sample of responses coded by a second coder. Inter-coder reliabilities (κ) for each theme are indicated.

*** denotes high apparent reliability due to low occurrence of coded measure (i.e., too few affirmative counts).**

Theme	κ	Values / Description	Example Participant Text
Inputs	.95	Sensor thresholds, ranges, sensitivity, limitations, capabilities, coverage, etc	“I would like to know what triggers the alarm” “Where does it get its forecasting information? How did it know I was leaving rather than entering my house?”
Model	.97	Application conceptual model / Criteria / Heuristic / Rules / Logic / How does it know / how it works	*(See below)*
Outputs	1*	Options / Alternatives that the system could produce. *Distinguish from What Else*	“I would like to know fully what type of things that it would be able to detect *[recognition output]*” “I would want to know if there was a quicker route.”

Theme	κ	Values / Description	Example Participant Text
Why	1	*Why* did the application do X?	“Why did it let *[the message]* through?” “How does it know it is going to rain?”
Why Not	.96	*Why* did it *not* do Y?	“I would like to know why the application did not filter out this message.” “Why application failed to sense that I was free in the office and failed to trigger.”
What Else	.95	*What (else)* is it doing?	“What other message are in the queue” “My location, weather, and time.” “How I could look up more specific details about the event.”
How	.97	*How* (under what condition) does it do Y?	“I would be interested in knowing how it decides how long to suppress messages.”
What If	1*	*What if* there is a change in conditions, what would happen?	“I'd like to know in what will application do in certain situations”
Certainty	1*	Application confidence of its actions / decisions	“How can I be sure it is accurate?” “I'd like to know how accurate the chosen forecasting system was.”
Control	1*	How to change settings / thresholds	“I would want to know if I could put a time limit on it.” “I would like to know how to make it send the correct reminder.”

Table 5.2 (Continued).

First, depending on the situation, participants may or may not have any *information demand*. If they want more information, they may want to know more about how the application works, in terms of functional *inputs*, *outputs* and conceptual *model*. A conceptual model describes the decision and action processes performed by the application to use the inputs in producing output. Regarding the conceptual model, the participant may ask for information that can be represented in the following questions:

1. **Why** did the application do X?
2. **Why Not**: *why* did it *not* do Y?
3. **How** (under what condition) does it do Y?
4. **What (Else)** is it doing?
5. **What If** there is a change in conditions, what would happen?

Why and Why Not questions seek situation-specific information, while How seeks more comprehensive information of how the output can be obtained. Although context-aware applications that sense and act implicitly may elicit questions on what the application is doing, question 4 (What Else) does not ask this. Instead, participants were interested in knowing what else the application was doing (*e.g.*, whether the Digital Family Portrait had contacted emergency services after detecting the elderly family member had fallen). Regarding non-functional information about the application, users may want to know how certain the application is of its actions, decisions, and inferences (Certainty). In agreement with past research [Barkhuus and Dey, 2003b; Bellotti and Edwards, 2001; Dey *et al*., 2006], users also want to be able to Control the application (*e.g.*, change settings for reminders), and want information on how to control it. In addition to more information about the application, the user may also want to know more about the current Situation. Regarding the situation, participants expressed their questions mostly in terms of What Else, and sometimes as Why or Why Not.

We also used the survey responses to derive themes describing circumstances that *moderate demand* for these explanation types:

Application Satisfaction. (7-pt Likert scale) How satisfied the user is with the application behavior.

Impression. Coded response of whether the user has a *positive*, *neutral*, or *negative* impression of the application, given the situation.

Trust / Reliance. Coded response of whether the user is *satisfied* (believes application, trusts it) or *dissatisfied* (doubtful, lost trust, in disbelief, found fault, will abandon) with the application.

These measures were found to be correlated and so we combined them into a summed and reversed measure of application **Inappropriateness** (α=.68). We split the scenarios into 2 groups: those with a high or low Inappropriateness score (using a Tukey pair-wise test; p<.001).

Based on the application functionality in various scenarios, we developed more moderators. For example, because CybreMinder supports goals of pre-planned tasks, while the other three applications do not, scenarios can be separated into whether they are goal-supportive or not. The other moderators are:

Criticality. Whether the situation presented is critical. Situations involving accidents or medical concerns with the Digital Family Portrait and work-related urgency for the IM Auto-Notification were considered highly critical. Due to the profound influence of the high criticality of the fall and incontinence scenarios in the Digital Family Portrait survey, these scenarios were excluded from consideration of the other moderators.

Goal-Supportive. Whether the situation is motivated by a goal the user has (CybreMinder scenarios only).

Recommendation. Whether the application is recommending information for the user to follow or ignore (CyberGuide scenarios only).

Externalities. Whether the application is perceived to have high external dependencies (*e.g.*, getting weather information from a weather radio station) *vs.* being perceived as “self-contained.” CybreMinder and CyberGuide had perceived high external dependencies.

5.4.3 RESULTS: USER DEMAND OF INFORMATION TYPES

Results are shown in Table 5.3, describing the overall demand for various types of information, and how the demand changes due to moderating circumstances. Discussion is deferred until after we present the results of Experiment 2.

	Average (%)	Inappropriateness	Criticality	Goal-Supportive	Recommendation	Externalities
Information	72.5	↑↑			↑	↑↑
Application	59.8	↑	↓↓		↑↑	↑↑
Situation	12.0		↑↑	↑*	↓	
Inputs	19.8		↓		↑↑	↑↑
Model	33.6	↑↑		↑		
Outputs	2.9			↓	↑↑	
Why	19.0	↑				
Why Not	8.2	↑↑		↑↑		↑↑
How	13					
What If	0.5					
What Else	6.4					
Certainty	1.7					
Control	10.0	↑				

Table 5.3: Results of Experiment 1. The left column shows the percentage of participants who had a demand for various explanation types. The right columns show the effect size of whether higher moderator rating values (of each column) leads to increased (up arrows) or decreased demand. All results indicate Bonferroni-corrected (n=78) significant differences (p<.01; * denotes p<.05). Cohen's d is reported to determine the size of differences rather than just whether the differences are significant. Single arrow indicates small effect size $(.2 < |d| \leq .3)$, double arrows indicates medium effect size $(.3 < |d| \leq .8)$.

How to read: *e.g.*, 72.5% of participants demand information, in general; participants demand more information about why not when the application behavior is more Inappropriate.

5.5 EXPERIMENT 2: ASSESSING DEMAND FOR EXPLANATION TYPES

While Experiment 1 sought to elicit the types of information users wanted for explanations of context-aware applications under various moderating circumstances, Experiment 2 solicits user demand for the types of information through explicit suggestion of questions and provision of explanations and studies the impact of meeting this demand on user satisfaction. For each of the explanation types identified in Experiment 1, we wrote corresponding questions and explanations for each application situation response of Experiment 1. We added one more explanation type in response to demand for more information about the application conceptual model. This "Visualization" explanation type provides a diagrammatic representation of the underlying decision tree. Table 5.4 shows examples of questions and explanations. We prepared 11 explanation types for each of the 54 scenarios, for a total of 594 explanations; some are repeated for similar scenarios.

We hypothesize that: (i) when asked specifically about whether they want an explanation type (heretofore called *solicited information demand*), users should reflect the same demands (*elicited information demand*) as that of experiment 1; and providing information for demanded explanation type will (ii) increase application satisfaction, and (iii) increase user rating of that explanation type.

5.5.1 METHOD

We reused the applications and scenarios from Experiment 1, with one application per survey. Experiment 2 is designed as a between-subject study for the explanation types (11 conditions plus a None condition). Participants are assigned to a version of the survey with only one Explanation type provided (including None). To mitigate order effects, four versions of each survey were deployed with a different random ordering of scenarios. On the survey, participants use a 7-point Likert scale question to rate their satisfaction with the application. In the explanation type conditions, participants were provided with a corresponding intelligibility question and explanation. We posed additional queries regarding how important it was to get the question answered, how much they are satisfied with the explanation and how useful they found it. We recruited 610 participants (42% female; ages 18 to 61; M=28.9) from Amazon Mechanical Turk, and evenly distributed them across the 12 conditions. Participants were paid $2.

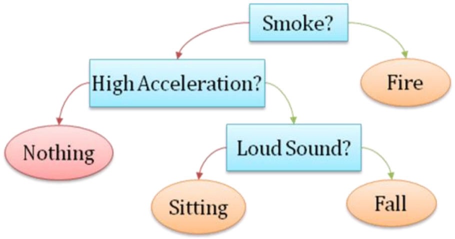
Explanation	Sample Question	Sample Explanation
Certainty	How certain is the system of this report?	The system is 90% certain of this report.
Inputs	What does the system use to sense accidents?	The system uses an accelerometer worn by the elderly family member, speakers around the home, and smoke detectors.
Outputs	What accidents can the system sense?	The system can sense the following accidents: Falls and Fire/Smoke.
Why	Why did the system report a fall?	The system reported a fall because there was a high acceleration from the accelerometer worn by the family member, and there was a loud sound.
How	How does the system distinguish a between a falling object and person?	The system did not report a fire, because the smoke detector did not set off.
Why Not	Why did the system not report a fire?	The system detects a fall through high acceleration detected from the accelerometer worn by the elderly family member, and a loud noise. The system detects a fall of an object through a loud noise, but no high acceleration from the accelerometer.
What If	If an object falls, would the system report a fall?	If an object falls, but the accelerometer worn by the family member does not report a high acceleration, the system would not report a fall.
What Else (Application)	Did the system alert emergency services of the accident?	The system has not alerted emergency services and is pending your approval.
Conceptual Tree Model Visualization	What is the overall model of how the system works?	The following diagram describes a simplified view of the conceptual model of how the system works. It works by tracing a decision tree and taking branches at decision points to arrive at a conclusion. Red arrows indicate false conditions, and green indicates true. For example, if smoke is detected, then the system concludes there is a fire.  Smoke? High Acceleration? Fire Nothing Loud Sound? Sitting Fall
Control	How can I change settings to control the sensitivity for reports?	Some settings can be changed through a control panel accessed via the menu: Options > Settings.
Situation (What Else)	What was the family member doing before the accident?	The family member was preparing dinner in the kitchen.

Table 5.4: Sample questions and explanations of various explanation types participants received for the Digital Family Portrait.

5.5.2 RESULTS ANALYSIS

We report three metrics in experiment 2: (i) *solicited intelligibility demand*, (ii) *satisfaction* of the application with and without intelligibility, and (iii) user rating of the *usefulness of intelligibility*. Table 5.5 summarizes the results of Experiment 2 and compares them with Experiment 1. Results are reported for explanation types across all scenarios, and by moderating circumstances (high *vs.* low rating groups) as described in Experiment 1.

5.5.2.1 SOLICITED INTELLIGIBILITY DEMAND (SID)

Participant solicited intelligibility demand is measured from responses to the 7-point Likert scale question on how important it is to receive an answer to the supplied explanation type question. All described differences are significant (p<.05) using a Tukey pair-wise comparison test. As with experiment 1, the moderating effects are measured by the effect size between low and high moderator rating groups.

5.5.2.2 DIFFERENCE IN APPLICATION SATISFACTION (ΔAS)

To calculate relative application satisfaction from None, for each scenario, we subtracted the means of satisfaction of the None condition from each response in the 11 explanation type conditions. Across moderating circumstances, we calculate the effect size between the high and low moderator groups. If the effect size is small or medium, we report the *difference* in scale value mean along with the p value of a t-test between the groups.

5.5.2.3 INTELLIGIBILITY USEFULNESS RATING (IUR)

We derived an intelligibility usefulness rating based on the mean of responses regarding explanation satisfaction and explanation usefulness (Cronbach's α=.84). We used the same analysis as described for *siD*.

	Average				Inappropriateness				Criticality				Goal-Supportive				Recommendation				Externalities			
	Exp1		Exp2		Exp1		Exp2		Exp1		Exp2		Exp1		Exp2		Exp1		Exp2		Exp1		Exp2	
	eiD(%)	*siD*	*ΔaS*	*iuR*	*eiD*	*siD*	*ΔaS*	*iuR*	*eiD*	*siD*	*ΔaS*	*iuR*	*eiD*	*siD*	*ΔaS*	*iuR*	*eiD*	*siD*	*ΔaS*	*iuR*	*eiD*	*siD*	*ΔaS*	*iuR*
Information	72.5	5.18	0.11	4.91	↑↑		0.47	-0.47		0.81		0.45					↑				↑↑			
Application	59.8	5.22	0.09	4.9	↑		0.45	-0.48	↓↓	0.79		0.44					↑↑				↑↑			
Situation	12.0	4.82	0.28	4.92			0.76		↑↑	1.15		0.62	↑*		-0.71	-0.75	↓							-0.52
Inputs	19.8	5.61	0.15	5.17			0.44*	-0.45	↓	0.50*				0.61			↑↑	-0.66			↑↑			
Model	33.6	5.04	0.01	4.7	↑↑		0.46	-0.52		0.83		0.41	↑	0.42				-0.45		-0.42				
Outputs	2.9	5.62	0.06	5.11			0.59			0.75		0.53	↓			-0.56	↑↑							-0.48*
Why	19.0	5.27	-0.36	4.32	↑			-0.65		1.21											0			0.59
Why Not	8.2	4.57	-0.01	4.42	↑↑		0.68			0.91			↑↑	0.95		0.74		-0.68			↑↑			0.44
How	13.4	5.20	0.22	5.18				-0.48				0.54		0.73				-0.67	-0.80	-1.11			-0.63	-0.75
What If	0.5	5.15	0.11	4.64			0.49			0.97								-1.33		-1.16		-0.84		-0.88
What Else	6.4	5.09	0.04	4.83				-0.83	↑	0.81														
Viz.		5.29	0.29	4.71		-0.37	0.53	-0.56		0.84		0.88			-0.62	-0.83			0.54					
Certainty	1.7	5.25	0.33	5.13				-0.66		0.99		0.42*		0.60					-0.49*	-0.51*		0.62		-0.43
Control	10.0	5.37	0.23	5.42	↑		0.79			0.54*		0.53				-0.80								-0.60

Table 5.5: Results of Experiment 2 and Experiment 1. eiD: elicited information demand. siD: solicited information demand. ΔaS: difference in application satisfaction for providing explanation type. iuR: intelligibility usefulness rating of explanation received. For Experiment 2, left columns under Average are the mean values of the Likert scale for siD and iuR, and ΔaS, the difference in means between intelligibility-provided (various types) and non-intelligible. The right columns are differences in means or differences between high and low moderator rating groups. All Experiment 2 results of each measure are Bonferroni corrected (n=84) and significant (p<.01, * denotes p<.05).

How to Read: *e.g.*, for explanation type Situation, 12% of participants in experiment 1 were interested in knowing about the situation; in experiment 2, participants wanted to know about it (M=4.82, above neutral); they rated it well, in general (M=4.92) and had a significant improvement in application satisfaction (ΔM=0.28). When considering scenarios according to Criticality, participants want more information about the situation (both elicited and solicited, both medium effects), experienced more satisfaction after receiving explanations for more critical scenarios (p<.05), and rated situation explanations better for those scenarios (small effect).

5.6 DISCUSSION

We discuss the meaning of the results from both experiments in terms of demand for intelligibility and what they imply for design. Among the four context-aware applications we investigated and their various scenarios, we have determined average demands for various explanation types. However, these demands change depending on circumstance and function of the application. Participant awareness to seek particular explanation types also increases their demand for these types. We also found that explanations should be carefully tailored to be effective, otherwise, they may be more detrimental than helpful.

Participants generally wanted information about the Application, and designers can anticipate this, but they also moderately wanted information about the Situation in which the application acts. When asking about the application, participants tend to focus on the application Model rather than the Inputs or Outputs. Of the application model, they care most about Why the application behaved as it did in specific situations, and How it works in general and How To produce certain actions or decisions. Participants indicated a strong demand for Visualizations (with significant increase in application satisfaction, and intelligibility explanation rating). This may be because participants prefer graphical to textual explanations, so visualization-based explanations may be a better means to deliver explanations. As in Chapter 4 [Lim, Dey, and Avrahami, 2009], Why Not explanations had a significant but limited demand compared to why. As indicated by many researchers (*e.g.*, [Barkhuus and Dey, 2003b; Bellotti and Edwards, 2001; Dey *et al.*, 2006]), our participants were interested in knowing how to *control* applications. We found that users were more satisfied with receiving this explanation type even though we only told them where they could find a control panel. This echoes findings in [Glass, McGuinness, and Wolverton, 2008] of how users adopt a “trust but verify” approach and just need to know they can override the settings, but not necessarily do so.

5.6.1 DEMAND FOR INTELLIGIBILITY VARIES WITH CIRCUMSTANCES

The Why Not explanation type is particularly effective for *Inappropriate* circumstances and Goal-Supportive functions. Participants would tend to ask Why Not when encountering an inappropriate behavior, especially if it deviates from their goals. Output information is desired more for *Recommenders* and Inputs information for applications with high *Externalities*. Table 5.5 indicates other explanation types that vary by circumstances.

5.6.2 HIGH INTELLIGIBILITY DEMAND FOR CRITICAL CIRCUMSTANCES

Criticality is a particularly significant moderating circumstance as for highly critical scenarios, participants want as much information about any explanation type (especially about the Situation and What Else) as they can get if they are aware of its availability. However, it is hard to satisfy participants with any explanation types in highly critical scenarios, possibly due to the stress during the critical period, and preoccupation with the primary problem.

5.6.3 AWARENESS OF EXPLANATION TYPES CHANGES DEMAND

Just as in [McGuinness *et al*., 2007], we found that awareness of the existence of explanation types plays an important factor where participants recognize the value of the various types and subsequently had greater demand for them, resulting in higher application satisfaction; this created higher demand and benefit than would have been predicted from Experiment 1. Though not in demand when elicited, Inputs, Outputs, What If, What Else, and Certainty became particularly desired when participants were informed or reminded about them.

5.6.4 VALLEY OF EXPECTATION

The elicited and solicited information demand measures indicate which explanation types are desired and under what circumstances, but providing explanations matching these types in the corresponding circumstance may lead to poorer intelligibility usefulness ratings instead. This is particularly evident for Inappropriateness circumstances, but absent for highly Critical ones. This suggests that when participants have a higher demand for intelligibility, they also expect better explanations, and may rate explanations worse. However, when they become more desperate for information (as in cases of high criticality), they readily accept the explanations, and rate them more favorably. We refer to this drop then rise in expectation of explanation quality with respect to information demand as the “valley of expectation” of intelligibility demand.

5.6.5 SATISFACTION *VS*. UNDERSTANDING

This work has explored the explanation types that users demand and that may satisfy them. We compare and contrast our findings to our earlier work in Chapter 4 that explored how to improve user understanding and trust of a context-aware intelligent system. Previously, we found that Why and Why Not explanations were most effective, while interactive explanation types (How To and What If), were less useful due to the difficulty of using them. In this study, we provide participants with various explanation types without involving any user interaction, so we avoid the confound of difficult user interfaces and, in fact, find that How (To) and What If explanations should not be neglected. We also found that even though Why and Why Not explanations are intuitive and particularly good at improving understanding, users have higher expectations on the informational quality of such explanations, and may be biased against appreciating them as much.

5.7 DESIGN RECOMMENDATIONS

We present some recommendations to developers of context-aware applications about which explanation types to provide and under what circumstances. Developers can identify which moderators and situations apply for their applications and then use the suggested recommendations. Implementing all the types of intelligibility we investigated is excessive and may even be detrimental. We present the different types and offer advice about when and how they should be implemented (summary in Table 5.6 and Table 5.7).

Application. When users want information, they will tend to want application-centric information, so, in general, more effort should be concentrated on providing information about the application's workings and mechanism than on the situation. Explanations about the application should also be provided for applications that are at risk of being easily abandoned (low trust), and when the application is uncertain (high risk of Inappropriateness) of its action.

Situation. Though users are less interested about what else is happening when the application responds, there is moderate interest in increasing their real-world situational awareness (*what else*). This would involve making applications sense more related events and contexts (*e.g.*, for a monitoring application, what is the historical trace of events before an anomaly) than their primary function, and to reveal such knowledge to the user. This is more important in cases when the application acts in highly critical situations.

Inputs. Users may only have a moderate interest in knowing more about the application's *input* sources or sensor readings, but if they perceive the application is heavily dependent on external sources (high externality), they may want to know more about the inputs.

Outputs. In typical application use, users are not interested in the output alternatives. However, they may suspect that the application is capable of more action than it is exhibiting. Providing intelligibility explaining the *output* (action) capabilities of the application is particularly important for applications that make recommendations (such as tour guides), especially when users want to seek better options. This should be provided automatically during early stages of usage to improve users' awareness.

Model. Most of the questions users want to ask are about an application's conceptual *model*. This is elaborated with the following explanation types.

Why. Answering *why* questions is an essential intelligibility requirement as such questions are very common. However, users may also have high expectations that the *why* explanations are very informative and a simple reason trace may not be sufficient to fully satisfy the users' enquiries. Visualizations could be used to augment a trace.

Why Not. Such explanations are good for high risk (high chance of inappropriateness) circumstances and goal-supportive functions. However, we caution against implementing it in all types of context-aware applications because generating these explanations for all alternative possibilities may be non-trivial.

How. Users are somewhat interested in knowing how the application arrives at its outcomes, and particularly like such explanations. However, *how* explanations can get cumbersome to produce for applications with complex or learned logic. Users may have to use an interactive facility to specify the constraints in which to obtain an action (as was the case in Section 4.6.4).

What If. This explanation type also involves user interaction in specifying input conditions and the application simulating what would happen. We recommend *what if* explanations for non-recommenders and more self-contained applications, for which users indicated strongest demand.

What Else. The *what else* explanation provides information closely related to the *situation* explanation. The latter provides situational awareness, while the former provides more information about what the application has done. Unsolicited demand for *what else* information is low, but becomes significant when participants are aware of its availability. This indicates an intrinsic need for this type of explanation. *What else* explanations are also important in critical situations when users hope that the application is doing more to remedy or handle the critical situation.

Visualization (Viz.). Given the general demand and effectiveness for this explanation type, we recommend providing visualizations to augment explanations.

Certainty. Providing *certainty* information is particularly important for applications that are goal-supportive where users want to know how certain the application is in its decision or action, and for applications that rely heavily on external sources and sensors. *Certainty* values that we provided for this study were over 90%, but we suspect that if low *certainty* accuracies are reported to the user, this may hurt their impression of the application. As applications can have varying levels of certainty when in use, it may not be wise to always show *certainty* information.

Control. For context-aware applications, this explanation type would support users changing parameters in the conceptual model that were originally set by the developers or learned by the system. However, when users adjust such parameters, they may be making poorer choices than the developers or the underlying machine learning algorithm, and may ultimately hurt the application accuracy. Nevertheless, it may be important to allow users to change these settings, but caution them about the danger of doing so.

5.7.1 DESIGN PRESCRIPTION

We synthesize our findings and discussion to produce an initial attempt at a design prescription on what and how to provide and implement explanation types for context-aware applications. We propose a four-step procedure, and use UbiGreen [Froehlich *et al*., 2009] as an example application to illustrate this and for external validity. UbiGreen is a mobile context-aware application that uses a wearable sensor to recognize physical activity relating to sustainability (green) actions. The application tracks the accumulation of green actions and displays a corresponding rewarding wallpaper on the phone throughout the week. The steps are as follows:

I. Map application to moderating circumstances. Determine whether the application will encounter high or low moderator rating values. Table 5.8 shows the mapping for UbiGreen.

II. Referring to Table 5.6 and Table 5.8, determine which explanation types to provide and prioritize them in order of recommendation.

III. Consider issues (such as obtrusiveness and privacy) that would lead to trade-offs with when and how to provide intelligibility. Table 5.9 lists concerns for UbiGreen, and Table 5.7 presents ways to provide explanation types.

IV. Summarize selections of explanation types with justifications. We provide Table 5.10 as a template for this summary and filled it out with details for UbiGreen.

		Explanation Type	General	Application Behabvior **Inappropriateness**		Situation **Criticality**		Application **Goal-Support Role**		Application **Recommender Role**		Number of Context **Externalities**	
				Low	High	Low	High	Low	High	Low	High	Low	High
Application		Inputs					✓						✓
		Outputs					✔				✓		
	Model	Why	✓	✓	✓	✓	✔						
		Why Not			✓		✓		✔				✓
		How	✓	✓	✓	✓	✓			✓		✓	
		What If					✓			✓		✓	
		What Else					✔						
		Visualization	✓	✓			✓	✓			✓		
		Certainty	✓				✔		✓				
		Control	✓		✓		✔						
Situation							✔						

✓ **=recommended,** ✔ **=highly recommended**

Table 5.6: Design prescription of which explanation types to implement depending on the circumstances encountered by and functionality of the candidate context-aware application.

Provision Type	**Tradeoffs**
Always on	Obtrusive and space consuming. Only suitable for displays that are persistent (e.g. peripheral or kiosk displays).
Automatic Context-Based	Could be obtrusive for frequent events. May want to deactivate after a while.
Adaptive / Intelligent	The system uses context to determine when to provide explanations. Applications with poor accuracy may provide or omit providing explanations at inappropriate times. This can be used to determine the most crucial time to send privacy-sensitive information.
On Demand$_a$ Always	Least obtrusive, but may not expose user frequently enough to improve their understanding.
On Demand$_e$ Context-Based	Allows users to get intelligibility features contextually.

Table 5.7: Reference of provision types to handle tradeoffs between providing intelligibility and other issues.

Moderator	Lo	Hi
Inappropriateness	✓	✓
Criticality	✓	
Goal-Supportive		✓
Recommendation	✓	
Externalities	✓	

Table 5.8: Circumstances mapping for UbiGreen.

Issues	Concern
Obtrusiveness	Med
Personal Privacy	Low

Table 5.9: Issues mapping for UbiGreen.

Explanation Type			Provision	Priority	Details / Comments
		Situation	-		
Application		Inputs	-		
		Outputs	-		
	Model	Why	On Demand$_{e}$	Med	To support curiosity or deal with expectation mismatches.
		Why Not	On Demand$_{a}$	High	Needed to answer why certain activities were not recognized and recorded.
		How	On Demand$_{e}$	Med	Providing this explanation type may be controversial, since this information allows users to game the system.
		What If	-		Implementing this will also support gaming of the system.
		What Else	-		
		Visualization	(with others)		To augment textual explanations.
		Certainty	Always	Med	Though important to have, it is more appropriate to show certainty at the event level (e.g. detection of various activities), than at the cumulative level (i.e. wall paper display of progress).
		Control	On Demand$_{a}$	Med	Expose user model and sensor thresholds to allow tweaking.

Table 5.10: Design prescription for UbiGreen.

5.8 LIMITATIONS

We acknowledge three limitations of our approach. First, we do not claim our list of moderating circumstances to be comprehensive; there are other explanation types that could be important to context-aware applications (*e.g.*, history, similar examples), and other circumstances (*e.g.*, more vs. less autonomous systems). Second, our participants had to imagine using the various applications through video and text. We used this approach to garner opinion from a larger pool of participants and to investigate a range of multiple context-aware applications. However, this does not mitigate the need to investigate the demand for intelligibility features in deployed systems, where users can get real-time feedback, have internal motivations and goals, and can draw on past experience of using the system. Third, since we drew our participants from the Internet, and a relatively new platform, Mechanical Turk, our population is not a representative sample of the general public (see [Schonlau *et al*., 2006]). Our sample (ethnicity 32% Asian / Pacific Islander, education 37% 4-year college educated, 18% with post-graduate degree; 27% students) is more representative of an internet-savvy population, and should be representative of technology early adopters. This population is appropriate, as we are dealing with the adoption of novel technologies.

5.9 CONCLUSIONS AND FURTHER WORK

We have described two experiments about a wide range of scenarios with four context-aware applications to assess user demand for intelligibility and the impact on user satisfaction when those demands are met. From a question asking perspective, we have elicited a set of explanation types (Application, Situation, Inputs, Outputs, Model, Why, Why Not, How, What If, What Else, Certainty, Control) users of context-aware systems may be interested in, and several circumstances that may moderate when this demand changes (*Application Behavior Inappropriateness*, *Situation Criticality*, *Application Goal-Supportiveness*, *Recommendation Role*, *Number of Externalities*). Our findings suggest that some explanation types (*e.g.*, Why, Certainty, Control) should be made available for all context-aware applications, while some are more useful for specific contexts (*e.g.*, Why Not for goal-supportive tasks). We believe that context-aware application developers can take these recommendations on when and how to provide different types of intelligibility features and dramatically improve user satisfaction with, and acceptance of, their context-aware applications.

We have found that even if certain explanation types are helpful to improve users' understanding, they may not necessarily want or seek them, and be more satisfied. We would like to investigate the interactions involved in increasing understanding and satisfying information demands to provide a more cohesive framework for intelligibility. We wish to test our findings in a field study with a deployed intelligible context-aware application. As an intermediate step, we designed, developed and evaluated, Laκsa, an intelligible context-aware mobile application (described in Chapters 7 and 9) as a research platform. This application provides 10 of these explanation types allowing us to gain richer insights into how and when users interact with each type, and how that may improve their understanding of the application behavior.

To support the implementation of Laκsa and other intelligible context-aware applications, and to make it easier to implement explanations that can explain the broad range of question types we introduced in this Chapter, we developed the Intelligibility Toolkit, which we describe in Chapter 6.

6 INTELLIGIBILITY TOOLKIT

This chapter is an extension of the work presented in:

> Lim, B. Y. and Dey, A. K. (2010). Toolkit to Support Intelligibility in Context-Aware Applications. In *Proceedings of the 12th ACM international Conference on Ubiquitous Computing (Copenhagen, Denmark, September 26 - 29, 2010). Ubicomp '10*. ACM, New York, NY, 13-22.

ABSTRACT. Context-aware applications should be intelligible so users can better understand how they work and improve their trust in them. However, providing intelligibility is non-trivial and requires the developer to understand how to generate explanations from application decision models. Furthermore, users need different types of explanations and this complicates the implementation of intelligibility. We have developed the Intelligibility Toolkit that makes it easy for application developers to obtain 12 types of explanations (*e.g.*, Certainty, Why, Why Not, What If, How To) from the several popular decision models (*e.g.*, rules, decision trees, naïve Bayes, hidden Markov models) of context-aware applications. We describe its extensible architecture, and the explanation generation algorithms we developed. We validate the usefulness of the toolkit with four canonical applications that use the toolkit to generate explanations for end-users.

6.1 INTRODUCTION

Having elicited what questions users are interested to ask of context-aware applications in Chapter 5, we have found a reasonably large set of explanation types that applications may provide to be intelligible. However, it would be a substantial effort for developers to implement these explanations for all context-aware applications for which they would like to provide intelligibility. To support the implementation of intelligibility, and lower the barrier to providing them in context-aware applications, we developed the Intelligibility Toolkit. Its focus is to provide the automatic generation and processing of explanations from the underlying context model of a context-aware application. The Intelligibility Toolkit provides an extensible, standardized framework with which the explanations can be (i) selected, (ii) queried, (iii) generated, (iv) post-processed, and (v) presented. Our contributions are:

1. An architecture for generating a wide range of 12 explanation types drawn from Chapter 5. Our current implementation extends the Context Toolkit, a popular toolkit for building context-aware applications [Dey, Abowd, and Salber, 2001] and supports at least 10 popular inference models (*e.g.*, rules, decision trees, naïve Bayes, hidden Markov models).
2. A library of reference implementations of explanation generation algorithms we developed to extract any of the explanation types from any of the inference models we support.
3. Automated support for the recommendations from Section 5.7 that promotes good design practice by making the most contextually appropriate explanations easy for developers to acquire. Applications can automatically obtain the most appropriate explanations given the contextual situation.

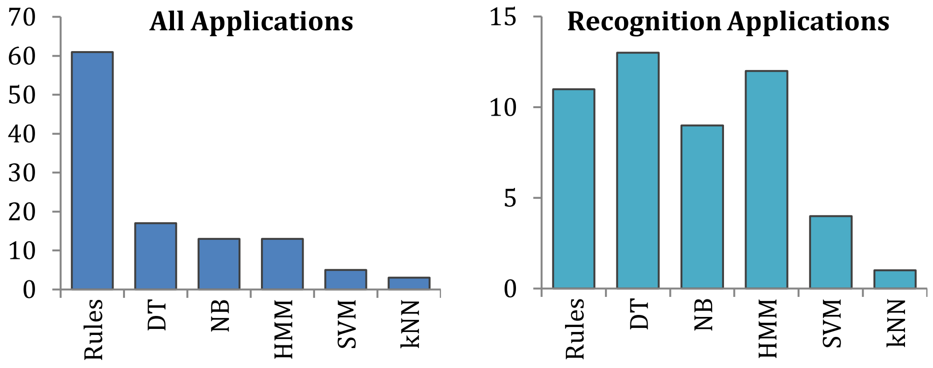
As we will show, these contributions satisfy the requirements laid out by Olsen [2007] for adding value to user interface architectures: (i) importance, (ii) problem not previously solved, (iii) has *generality* across a range of explanation types and decision model types, (iv) *reduces solution viscosity* through increased *flexibility* for rapid prototyping of explanations, (v) *empowers new design participants* by making it easier to provide explanations, and (vi) demonstrates *power in combination* by supporting combinations of explanation types as building blocks.

This chapter is organized as follows: we review context-aware applications published in recent years from a number of premier conferences to ascertain popular inference models. Then we present a discussion of the explanation we seek to support. Next, we provide an overview of the toolkit architecture and components, and implementation details and algorithms. We validate the toolkit through demonstration applications and the application of Olsen's infrastructure guidelines. We then discuss other toolkit features that would be valuable to support, and compare the toolkit with related work. We end with discussing opportunities for new research given the toolkit.

We have also included extensive appendices to this chapter, describing where the toolkit may be downloaded (Appendix A), detailing more Explainer explanation generation algorithms (Appendix B), and providing a tutorial on how to build an intelligible context-aware application with the Intelligibility Toolkit (Appendix C).

6.2 INFERENCE MODELS IN CONTEXT-AWARE APPLICATIONS

In [Lim and Dey, 2010], we identified the most popular context-aware inference models to support in the Intelligibility Toolkit. We reviewed literature from three major conferences over at least five years: CHI 2003-2009, Ubicomp 2004-2009, and Pervasive 2004-2009. We found the four most popular models among the 114 context-aware applications reviewed: rules, decision trees, naïve Bayes, and hidden Markov models (see Figure 6.1).



Key: Decision Tree (DT), Naïve Bayes (NB), Hidden Markov Models (HMM),
Support Vector Machines (SVM), k-Nearest Neighbor (kNN)

Figure 6.1: (Left) Counts of model types used in 109 of 114 reviewed context-aware applications from 2003-2009. (Right) Counts for 50 recognition applications; classifiers are used most often for applications that do recognition.

In this chapter, we extend the Intelligibility Toolkit in [Lim and Dey, 2010] to support more types of inference models to provide developers with even more flexibility in choosing machine learning classifiers and yet retain intelligibility features in the toolkit.

6.2.1 LOGIC RULES

We classify applications as rule-based if the authors state that their applications were based on Boolean rules (*e.g.*, if/else logic; AND and OR combinations), or that they were based on simple mapping associations of IDs to entities (*e.g.*, RFID). Developer specified rules are the most popular decision models used in context-aware applications. They are popular in the following domains: activity recognition (*e.g.*, [Tsukada *et al.*, 2004]), adaptation / personalization (*e.g.*, [Terada *et al.*, 2004]), awareness / monitoring (*e.g.*, [Dey and Guzman, 2006]), reminders (*e.g.*, [Borriello *et al.*, 2004]), location guides (*e.g.*, [Newcomb *et al.*, 2003]), and persuasion (*e.g.*, [Froehlich *et al.*, 2009]).

Also, the number of rule-based toolkits for context-aware applications indicates the popularity of rules (*e.g.*, [Assad *et al*., 2007; Bardam, 2005; Dey and Newberger, 2009; Gu *et al*. 2005]).

6.2.2 DECISION TREES

Decision tree classifiers (*e.g.*, C4.5 [Quinlan, 1993], Random Tree of a Random Forest [Breiman, 2001]) learn a decision tree from a dataset. A decision tree infers an output by deciding on a specific input feature at each node as it traverses down and returns a decision once it reaches a leaf. Decision trees are popular for their simplicity of use, interpretability, and good runtime performance. Decision trees are popular in applications to recognize: identity / ability (*e.g.*, [Chang *et al*., 2009]), interruptibility (*e.g.*, [Avrahami and Hudson, 2006; Tullio *et al*., 2007]), mobility (*e.g.*, [Zheng *et al*., 2008]), *etc*.

6.2.3 FUNCTIONS

A simple form of *functional relations* is the direct of one or more inputs to an output. For example, adjusting the color of an ambient display based on the current price of energy (Ambient Energy Orb[1]), or adjusting the length of a string based on the arrival times of buses (BusMobile [Mankoff *et al*., 2003]). Other than programmer-defined functions, functions can also be learned using machine learning algorithms.

Linear Regression learns a linear equation relating input features to a continuous-valued output. Linear regression has been used to model: physical energy expenditure from wearable sensors (*e.g.*, [Albinali *et al*., 2010]), step count prediction (*e.g.*, [Sohn *et al*., 2006]), and in combination with other classification methods for domestic utility inference (*e.g.*, HydroSense [Froehlich *et al*., 2009], GasSense[Cohn *et al*., 2010]).

Logistic Regression is similar to linear regression, but instead learns a sigmoid function for use in classification instead of regression. Logistic regression has been used to model: availability from a mobile phone (*e.g.*, [Rosenthal, Dey, and Veloso, 2011]), physical activity monitoring (*e.g*, [Sun, Zhang, and Li, 2011], and dietary activity recognition with wearable sensors (*e.g.*, [Amft and Tröster, 2008]).

Support Vector Machine (SVM) classifiers are a popular machine learning classifiers that use a maximum-margin hypothesis to learn a linear equation relating input features to classify a class

[1] Ambient Devices. http://www.ambientdevices.com/products/what-is-ambient. Retrieved 3 April 2012.

value. A popular method to train SVMs is Sequential Minimal Optimization (SMO) [Platt, 1998]. SVMs have been used to model: eye movement tracking for activity recognition (*e.g.*, [Bulling *et al*., 2009]), visual memory recall (*e.g.*, [Bulling and Roggen, 2011]) physical activity recognition (*e.g.*, [Murao *et al*., 2009]), and gesture recognition using power-lines (*e.g.*, [Cohn *et al*., 2011]).

6.2.4 BAYESIAN MODELS

Several inference models use Bayesian inference to determine the output given some input values. Generally, they use Bayesian probability to determine the posterior probability of an outcome value due to a prior probability and likelihoods due to some variables. These variables provide evidence for the inference.

Naïve Bayes is a probabilistic classifier that applies Bayes theorem to model the probability of the output of a system given the inputs. It applies a naïve assumption that features are conditionally independent of one another. Training and runtime performance are fast. Naïve Bayes classifiers have been used to recognize: physical activity (*e.g.*, [Chang *et al*., 2007]), domestic activity (*e.g.*, [Tapia *et al*., 2004]), interruptibility (*e.g.*, [Tullio *et al*., 2007]).

Hidden Markov models (HMM) [Rabiner, 1989] are Bayesian probabilistic classifiers that model the probability of a *sequence* of hidden states given a sequence of observations (input features with respect to time). First-order Markov models assume that only the previous state affects the next, and only the current state influences the current observation. HMMs have been used to model: physical (*e.g.*, [Chang *et al*., 2007]) and domestic activity (*e.g.*, [van Kasteren *et al*., 2008]), gaze (*e.g.*, [Bulling *et al*., 2008]).

Bayesian Networks (also called Bayes Net, BN) are more general Bayesian probabilistic classifiers that can model more dependencies between variables in the model. BNs have been used to model: family transportation coordination [Davidoff *et al*., 2011], activity tracking of multiple home occupants (*e.g.*, [Wilson and Atkeson, 2005]), medication prompting (*e.g.*, [Vurgun, Philipose, and Pave, 2009]), and social sensing for epidemiological behavior change [Madan *et al*., 2010]. Explaining BNs have been extensively researched especially in the medical informatics literature (*e.g.*, see [Lacave and Díez, 2002]). As such, we do not currently support explaining BNs in the Intelligibility Toolkit.

6.2.5 SIMILARITY

Another approach to inferring an output value is to determine how similar the current inputs are to previously seen examples. This is a common approach for recommender systems that recommend based on items in which the user had expressed interest, and case-based reasoning systems, which refer to specific cases. We describe two common similarity-based models.

k-Nearest Neighbors (kNN) is an instance-based learning method that classifies outcomes based on the majority class value of the instances in the training set that are nearest to the test instance. kNN has been used to model: context-based mobile search [Lane *et al.*, 2010], single-point electronic device sensing [Gupta, Reynolds, and Patel, 2010], EEG-based attention recognition [Li *et al.*, 2011], and physical activity recognition [Bicocchi, Mamei, and Zambonelli, 2010].

k-Means Clustering is an unsupervised learning method to group data instances into *k* clusters based on how similar the instances are based on their input features. On its own, clustering does not classify a label or output value. However, k-means has been used together with supervised learning classifiers to model, *e.g.*, smart phone energy consumption [Oliver and Keshav, 2011]. We focus on supervised instead of unsupervised methods, and do not currently support explaining k-means clustering in the Intelligibility Toolkit.

6.2.6 ENSEMBLE INFERENCE MODELS

Ensemble meta-classifiers are commonly used to improve the accuracy of classifiers. We describe two popular ones.

Bootstrap Aggregation (also called Bagging) [Breiman, 1996] resamples a training dataset with replacement to train multiple versions of a base classifier, which subsequently are used to vote on an inference. Bagging has been used to model: cough detection using microphones [Larson *et al.*, 2011], cooperative people-centric inference from data of multiple people [Lane *et al.*, 2009], and transportation mode [Zheng *et al.*, 2010].

Adaptive Boosting (AdaBoost) [Freud and Shapire, 1995] is a popular Boosting method that iterates a base classifier multiple times by updating the training dataset of each iterated classifier to emphasize wrongly classified instances. Inference is performed through a weighted vote of the classifier iterations. AdaBoost has been used to model: physical activity recognition with boosted decision stumps (*e.g.*, [Lester, Choudhury, and Borriello, 2009]), hand gesture recognition with boosted kNN (*e.g.*, [Fukui *et al*., 2011]), mobility mode inference with boosted logistic regression (*e.g.*, [Sohn *et al*., 2006]).

Several applications also combine rules for higher level logic with classifiers for lower level recognition (*e.g.*, UbiFit [Consolvo *et al*., 2008], UbiGreen [Froelich *et al*., 2009], Laκsa [Lim and Dey, 2011a]).

6.3 EXPLANATION QUESTION TYPES

As we focus on these four decision models, we need to generate the various explanations that users want to receive. In Chapter 5 we enumerated ten explanation types that are important to end-users of context-aware applications, and in this work we provide mechanisms to *automatically* extract eight of them from the applications. We review the different explanation types and their definitions.

6.3.1 SYSTEM-BASED EXPLANATION TYPES

If we represent a context-aware application as sensing inputs, maintaining system state, and producing an output, we can identify a set of explanations that are independent of the decision model and how it makes its decisions.

1. **What** explanations inform users of the current (or previous) system state in terms of output value; this makes the application state explicit. If the application performs multiple actions per state, the user may ask **What Else** to learn of other actions or services that the application may have performed simultaneously.
2. **When** explanations inform users when the inference made, *i.e.*, the time at which the context changed its value.
3. **Inputs** explanations inform users what input sensors (*e.g.*, thermostat, GPS coordinates) and information sources (*e.g.*, weather forecast, restaurant reviews website) that the application employs so that users can understand its scope. Input values are obtained by recursively asking What on the Inputs. When a user asks a why question, she may naively be asking for the Inputs state. Inputs should be described by their name and possibly also with some description of what they mean or refer to.
4. **Outputs** explanations inform users what output options the application can produce. This lets users know what it can do or what states it can be in (*e.g.*, activity recognized as one of three options: sitting, standing, walking). This helps users understand the extent of the

application's capabilities. Outputs explanations can also be used to help ask model-based explanations Why Not and How To (see below) by allowing the user to select an alternative desired output.

5. **What If** explanations allow users to speculate what the application will do given a set of user-set input values.

6.3.2 MODEL-BASED EXPLANATION TYPES

Explanations regarding the inference mechanism of the decision making process in the application are model-dependent, and would vary depending on the model used.

6. **Certainty** explanations inform users how (un)certain the application is of the output value produced. They help the user determine how much to trust the output value.
7. **Why** explanations inform users why the application derived its output value from the current (or previous) input values. For rule-based systems, this returns the conditions (rules) that were true such that the output was selected.

   In terms of model generated explanations, a Why question asks for the reasoning trace(s) for the rule(s) that triggered the inferred value, or evidence for why the inferred value was inferred over alternative values
8. **Why Not** explanations inform users why an alternative output value was *not* produced given the current input values. They could provide users with enough information to achieve the alternative output value, but not necessarily so (*e.g.*, the Weights of Evidence style of explanation; see Section 6.4.2).

   In terms of model generated explanations, a Why Not question asks for a *pairwise comparison* between the inferred output value and an alternative output value.
9. **How To** explanations answer the question "In *general*, how can the application produce desired output value X?" This contrasts with Why that asks in regards to a specific event.

   Occasionally, providing a comprehensive How To explanation is infeasible, since there could be infinite solutions. There are other variants of How To questions that can provide a finite and tractable number of reasons, such as How To If, where the Input state is constrained when asking How To.

6.3.3 APPLICATION-BASED EXPLANATION TYPES

The following explanation types depend on the application and sensed or inferred context and are thus application-dependent.

10. **Situation** explanations seek to demonstrate or illustrate what was happening to provide a ground truth of what was being inferred, *e.g.*, playing an audio clip of what was heard to substantiate the application's sound inference. The manifestation of this explanation will depend on the physical context being explained.
11. **Description** explanations provide textual descriptions for teaching *terminology* and concepts of terms, features, or feature values [Swartout and Smoliar, 1987]. With longer text, description explanations can also support the *design rationale* explanation type of Haynes *et al*. [2009] and *justification* explanation type of Swartout [1983].

We omit explaining Control because this is already covered by Enactor Parameters [Dey and Newberger, 2009]. Description explanations may also be used to explain how to Control the application.

6.3.4 HISTORY EXPLANATION

History explanations inform users about the application state at a previous time. This is not truly a distinct explanation type, but a means to retrieve historical explanations of the aforementioned explanation types. Rather than explaining about the current inference, History explanations allow the user to learn about a past historical inference. Therefore, this may simply provide a historical What explanation, or provide a means to ask other questions and obtain explanations (*e.g.*, Why, Why Not, Inputs) about that time instance.

To support History explanations, we require the application to store and retrieve historical information at the application level or deeper at the system level. Furthermore, assuming that the inference models are static, *i.e.*, do not change over time, Historical explanations can then be generated by applying Explainers to the inference model with the specific historical state. If the inference model changes over time, then historical versions of the model will also need to be stored and retrieved to generate the corresponding explanation of the historical event.

Currently, the Intelligibility Toolkit does not specify any storage mechanisms, though the Context Toolkit does support storage of context information. We have developed an intelligible application prototype, Laκsa (see later Sections 7.2 and 9.3), which also supports storing and retrieving of historical contexts, and so supports historical explanation types.

6.4 EXPLANATION STYLES OF MODEL-BASED EXPLANATIONS

The differences in inference models affect how an explanation may be generated. We support two styles of explanations in the Intelligibility Toolkit that can be generated: rule traces and weights of evidence.

6.4.1 RULE TRACES

Rule traces pertain to the trace or line reasoning content type of explanation described in [Gregor and Benbasat, 2001]. They trace the line of reasoning that was executed to explain why an inference was made. Conveying traces that were not executed explain why an alternative inference was not made. For a logic inference model, a trace is just a conjunction (AND) of logic conditions. The Intelligibility Toolkit provides Rule Trace explanations for Rules and Decision Tree inference models.

6.4.2 WEIGHTS OF EVIDENCE

Many models do not make inferences using rules (*e.g.*, naïve Bayes, SVM) and so rule traces are not relevant for explaining them. Instead, we employ the Weights of Evidence concept also used in [Kulesza *et al*., 2009; Poulin *et al.*, 2006; Madigan, Mosurski, and Almond, 1996; Mozina *et al*., 2004]. This considers that the model computes a *total* evidence for each possible outcome value that may be inferred, and that this total evidence is due to a sum of *atomic* weights of evidence due to various factors. In the inference models we consider for the Intelligibility Toolkit, we consider input features (*e.g.*, sound volume, frequency) as the main factors. Therefore, the weights of evidence explanation attempts to inform the user how much evidence each factor contributes towards or against the inference.

6.5 REQUIREMENTS

We seek to provide an Intelligibility Toolkit that supports the automatic generation of the aforementioned explanation types from various inference models. Context-aware applications built with the toolkit freely receive the capability of providing these generated explanations to end-users.

The toolkit is designed to satisfy *requirements* inspired from Olsen's writings about infrastructure evaluation [Olsen, 2007]:

R1) LOWER BARRIER TO PROVIDING EXPLANATIONS

With the toolkit, application developers do not need to know how to generate explanations from the most popular models used in context-aware applications. Explanation generation algorithms and heuristics are encapsulated into the toolkit.

R2) FLEXIBILITY OF USING EXPLANATIONS

Given the simplicity of invoking various explanation types, all types can be generated with the same level of ease. Developers can then concentrate on choosing the most suitable explanation for their applications and users. This supports rapid prototyping of providing explanation solutions to see which works best.

R3) FACILITATE APPROPRIATE EXPLANATIONS AUTOMATICALLY

Even with this rapid prototyping support, we encourage the use of appropriate explanations, particularly following the recommendations from Section 5.7 of how different explanation types are more appropriate in various contexts.

R4) SUPPORT COMBINING OF EXPLANATIONS

Some explanation types depend on other types to give a complete explanation to the user. For example, a Why Not explanation needs to inform the user of the set of Output values so that she would ask only about what is possible in the application. Other than nesting explanations, explanations can also be combined to enhance user experience. For example, combining the How To and What If explanations can expedite users in finding good examples to learn how an application works (e.g., see Section 6.10.6.3).

R5) GENERALIZABILITY

Although the toolkit currently covers a range of eight explanation types, four popular decision models and methods for simplifying and presenting explanations, like any toolkit, it is not comprehensive. The toolkit can be extended to support:

a) New explanation types and new methods to explain the supported models (*e.g.*, competing methods to explain naïve Bayes are presented in [Mozina *et al*., 2004; Poulin *et al*. 2006; Robnik-Šikonja and Kononenko, 2008]),

b) New algorithms for explaining currently unsupported inference model types (*e.g.*, SVM, clustering),

c) New heuristics for post-processing explanations,

d) New presentation styles for providing explanations, and

e) New heuristics for selecting explanations to provide.

6.6 TOOLKIT ARCHITECTURE

The Intelligibility Toolkit consists of four types of functional components, and three types of structural components. The functional components implement processes for generating explanations (Explainer), simplifying or reducing them (Reducer), and rendering or presenting them (Presenter). The structural components support the encapsulation and transport of explanations (Explanation Expression), and questions or querying (Query). Figure 6.2 shows how these components relate to one another in a pipeline to produce explanations for end-users to consume.

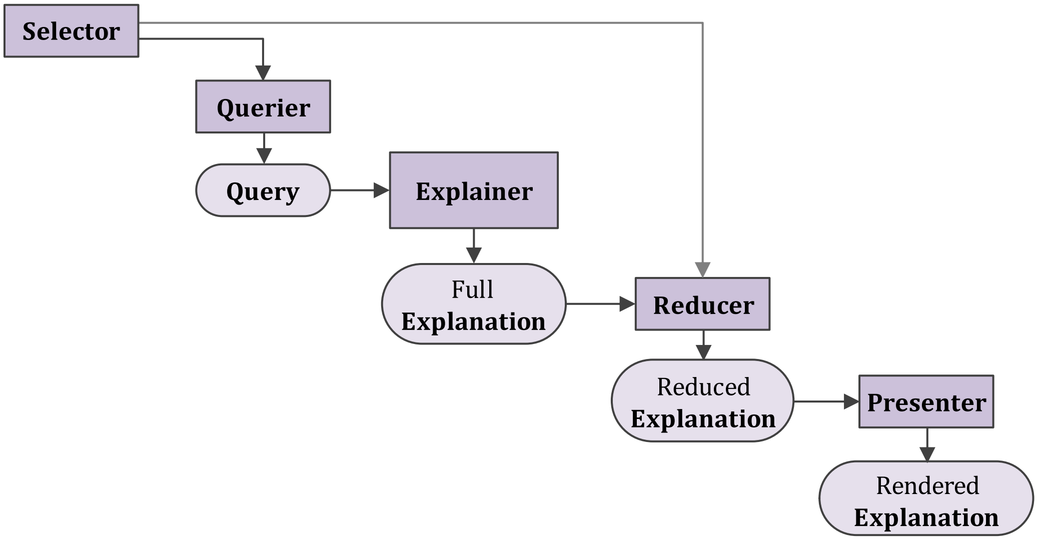


Figure 6.2: Architecture of the Intelligibility Toolkit showing functional and structural components.

To provide system-based explanations of context inference, we implemented the Intelligibility Toolkit, extending the Enactor framework [Dey and Newberger, 2009] of the Context Toolkit [Dey, Abowd, and Salber, 2001]. Enactors contain the application logic of the context-aware application and contain References that monitor the state of input Widgets.

For the rest of this section, we describe the components of the Intelligibility Toolkit, focusing on how to use them as *library programmers* (*i.e.*, programmers who will use the Intelligibility Toolkit components to make their applications intelligible [Hudson, 1997]). In later sections, we will describe how *toolkit programmers* can extend the Intelligibility Toolkit to develop their own components.

6.6.1 QUERY

To have an explainer generate an explanation, the program can pass a `Query` to the Explainer. Queries encapsulate information about which *context* the user is interested to ask about, which *question* the user is asking, and about which *time* the user is asking. The queries may be about

The base `Query` takes the current inputs and output values and is used for explanation types What, Why, and Certainty. `AltQuery` extends `Query` to include an alternative target output value to facilitate explanation types Why Not, and How To. `InputsQuery` extends `Query` to allow the setting of input values, supporting What If explanations. `Query` can be extended to employ different constraining mechanisms, such as querying based on time.

6.6.2 EXPLAINER

The Explainer is the main component of the Intelligibility Toolkit that contains the mechanisms and algorithms to generate explanations based on the inference model. There is a generic `Explainer` that generates system-based explanations types, and subclasses of Explainer for each of the inference model supported. System-based explanations are generated using the architecture of the Enactor framework to provide the corresponding information. Model-dependent explanations are generated from different Explainers for each model. Each Explainer takes a Query and the application state as input, and generates the explanation as an Explanation Expression data structure. Given the two styles of explanations supported in the Intelligibility Toolkit, we provide Explainers for Rule Traces and for Weights of Evidence. See Section 6.10 for some explanation generation algorithms.

6.6.3 EXPLANATION STYLES AND DATA STRUCTURES

Explanations are generated in a standard format so that they can be reused in other components of the Intelligibility Toolkit. However, they differ depending on the explanation style.

6.6.3.1 RULE TRACES IN NORMAL FORM

We define a rule trace explanation in terms of one or multiple reasons (*e.g.*, multiple reasons for Why Not). Each reason can be a singular conditional (*e.g.*, one certainty value for a Certainty explanation) or a conjunction (*e.g.*, multiple conditionals for a Why explanation). The conditional is the atomic unit of an explanation (*e.g.*, certainty $= 90\%$, temperature < 24°C). Furthermore, there can be negated condition literals (*e.g.*, $\neg$(temperature ≥ 24°C)).

Formally, for standardization, we define explanations in Disjunctive Normal Form (DNF), *i.e.* a *disjunction* (OR, $\cup$) of *conjunctions* (AND, $\cap$) of condition *literals* (see example in Figure 6.6, Right), or Conjunctive Normal Form (CNF), *i.e.* a conjunction of disjunctions of condition literals (see example in Figure 6.6, Middle). The standardization of explanation information supports Requirement R4 such that there is a consistent way to pipe different explanation types to other components in the toolkit.

6.6.3.2 WEIGHTS OF EVIDENCE AS A MULTI-DIMENSIONAL ARRAY

For a simple inference model, we can attribute the total evidence for an inference as a sum of each atomic weight of evidence due to an input feature. Note that the evidence is a continuous numeric value, while the input feature value may be nominal or numeric. This set of evidence just requires a one-dimensional array to represent the sum of weights.

However, some more complex inference models may have more factors that may ascribe evidence for the inference. For example, the HMMs also model time sequence, so weights of evidence will be due to input features over a range of time steps. In this case, we have time as a second dimension. Another example is using an ensemble meta-classifier that has multiple base classifiers, each performing an inference before the meta-level classification. In that case, we have classifier as the second dimension. Explanations of these inference models will require a two-dimensional array to represent the sum of weights of evidence.

In general, there may be any number of dimensions for the weights of evidence, so we represent it using a multi-dimensional array (also called multi-array). The sum of weights gives the total evidence for the explanation, and each element gives an atomic unit of evidence. This is beneficial to using a one-dimensional array as we do not lose information from each dimension.

6.6.4 REDUCER

Explanations generated from Explainers may be unwieldy to an end-user and lead to information overload. Reducers simplify the explanation data structure so that the explanation is easier for users to interpret. Reducers can also be used to post-process explanations before presenting them, *e.g.*, stripping privacy-sensitive information, when explaining contexts to a social contact. Given the two styles of explanations the Intelligibility Toolkit has two kinds of reducers, one for rule traces (see Section 6.11.1) and another for weights of evidence (see Section 6.11.2).

6.6.5 PRESENTER

Explainers produce explanations in the form of Explanation Data Structures, and `Presenter`s render them in a form presentable to end-users, *e.g.*, as text, visualization, or interactive graphical interface. Developers can build different Presenters to suit their target user and device form factor. Furthermore, even if the explanation is large, a developer may elegantly present it (*e.g.*, see Figure 6.12), instead of reducing it.

6.6.6 QUERIER

`Queriers` are to `Queries` what `Presenters` are to `Explanations`: they provide interfaces to display questions that end-users may ask. This may involve using text input, drop-down menus, buttons, *etc*., and specific implementations also depend on the platform the application is deployed on, desktop, mobile, *etc*.

Rather than let end-users manually ask for explanations *on demand*, it may be useful to *automatically* provide the explanations (see Table 5.7). Queriers that show explainers automatically do not need to provide any user interface but just generate Queries to pass to Explainers.

6.6.7 SELECTOR

While the application may provide a range of explanations using the Intelligibility Toolkit, it may not be appropriate or helpful to provide all explanations all the time. As we will show in Chapter 8, providing explanations when the application is uncertain can harm a user's impression of the application. We have also seen how users prefer to ask different question types under various circumstances (Chapter 5). Therefore, it is important to allow the application to *selectively* provide explanations under appropriate conditions. In fact, this contextual selectivity of intelligibility is a specialized form of context-awareness. The Intelligibility Toolkit provides Selectors to support this need for *context-dependent intelligibility*.

Selectors take contextual factors as inputs, and selects appropriate explanation types that should be provided. They may select none, one, or multiple explanation types (*i.e.*, any number). When selecting multiple explanation types, they may also rank the selections to prioritize them. To facilitate automatic provision of explanations, Selectors may

Selectors may also be extended to select Reducers to apply after Explainers generate explanations. This way the application can control the level of detail to provide in explanations. This can facilitate showing more explanation detail when useful (see Chapter 9), and showing less explanation detail when too overwhelming to end-users (see Sections 7.3.1 and 7.10.1).

6.7 IMPLEMENTATION OF THE INTELLIGIBILITY TOOLKIT

We implemented the Intelligibility Toolkit in Java (JDK 7) with extensive use of generics. The Intelligibility Toolkit does not perform inference on its own; its main purpose is to explain the inference. Hence, it depends on third-party libraries for inference models. The Explainer component depends on the inference model being explained. System-based explanations depend on the system infrastructure for the inference. We implemented base Explainers for the Enactor framework and also for the WEKA machine learning toolkit [Hall *et al.*, 2009]. We implemented a Rule explainer to explain rules in Enactors, and several explainers to explain classifications in WEKA classifiers.

The Intelligibility Toolkit also does not enforce any particular user interface framework or platform, and focuses on explanation generation. Therefore, specific implementations of the Presenter component depend on the platform on which the context-aware application is deployed. We have implemented Presenters for desktop graphical user interfaces (GUI) using the Java Swing API, and for mobile applications using the Android 2.2 API.

The Intelligibility Toolkit library binary and source code is downloadable from http://www.contexttoolkit.org. We have also provided documentation and tutorials for developers.

In the next few sections, we describe of how the Intelligibility Toolkit is integrated with the Context Toolkit, implementation details of its core components, and how to extend each type of component.

6.7.1 EXTENDING THE ENACTORS FRAMEWORK OF THE CONTEXT TOOLKIT

The Context Toolkit was originally developed by Dey and colleagues to support the development of context-aware applications by abstracting the connectivity issues of devices in a distributed environment through the use of a *discovery* mechanism (blackboard) and by developing a *widget* paradigm for context-aware applications to abstract the building of context-awareness functionality and their use. We added support for machine learning and an XML-based method to define widgets and enactors. See documentation at http://www.contexttoolkit.org for more details. In this section, we describe some of the components of the Context Toolkit necessary to know how we have extended it for use in the Intelligibility Toolkit.

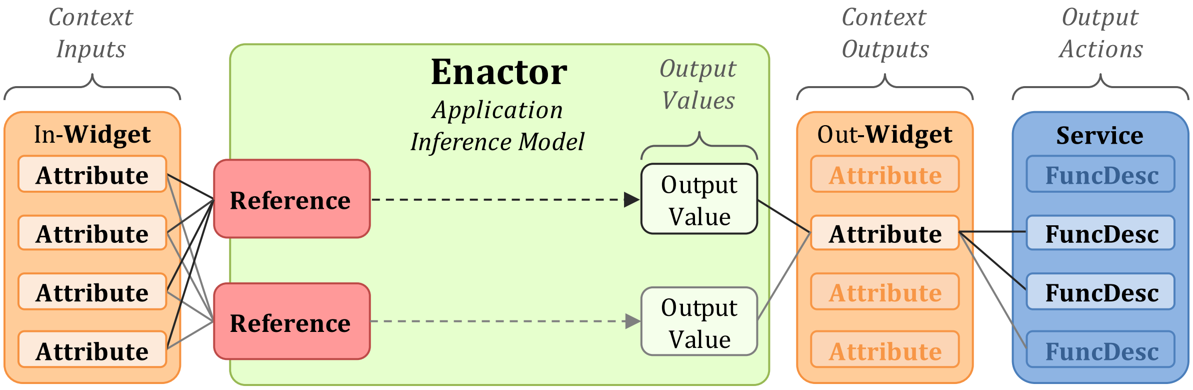


Figure 6.3: Architecture of Enactor framework of the Context Toolkit updated to include output values and In and Out Widgets.

6.7.1.1 WIDGETS

Context `Widgets` are responsible for separating the details of sensing context from actually using it. They abstract away the details of how the context is sensed from applications and other context components that need the context. Widgets here can be thought of as similar to GUI widgets (*e.g.*, text fields, spinners), but applied to sensors and actuators that would be common in ubiquitous computing systems. Each context widget is responsible for some small set of contexts that is captured from a (hardware or software) sensor.

Widgets have `Attributes` to represent contextual information. An `Attribute` represents a type of context, containing the name of the context and the data type of the context (float, int, string, etc).

Attributes are used to describe context components, and to specify what context a component is interested in. `AttributeNameValue` extend Attribute to also encapsulate the value of the attribute.

There are also Interpreters and Aggregators, but we leave it to the diligent reader to read the original papers describing them (*e.g.*, [Dey, 2000; Dey, Abowd, and Salber, 2001]).

6.7.1.2 SERVICES

While widgets contain information about context state, they can also encode behavior, through `Services`. Each `Service` object contains a service name and a `FunctionDescriptions` object that describes what the service does.

6.7.1.3 ENACTORS

Introduced into the Context Toolkit by Dey and Newberger [2009], `Enactors` encapsulate the application logic of context-aware applications. They contain `EnactorReferences` that specify rules for matching the state of input Widgets. Each `EnactorReference` contains a rule and is triggered when its rule is satisfied. This is managed automatically by the discovery mechanism of the Context Toolkit. Enactors provide basic support for Intelligibility (with `EnactorListeners`) and Control (with `EnactorParameters`).

We added an output property and a list of its *output values* for the Enactor to represent its output value, and output options. Each output value is associated with a Reference. Furthermore, while not necessary, we chose to delegate all inference mechanisms only in Enactors and leave Widgets as components for maintaining context state. Hence, for an Enactor, there is an *In-Widget* and an *Out-Widget*. Inference is performed over the attribute values of the In-Widget, and the inferred output value is assigned to the attributes of the Out-Widget.

6.7.1.4 GENERATORS

In some cases, there may not be an explicit In-Widget, such as when a raw signal is retrieved from an accelerometer. We introduce the Generator component as a one-sided Enactor that only has an Out-Widget. A Generator can be used to maintain the states of widgets using application-specific code. They can be thought of as “black box” enactors that do not have a defined input widget. The output widget gets its state updated.

6.7.2 ADDING SUPPORT FOR MACHINE LEARNING CLASSIFICATION

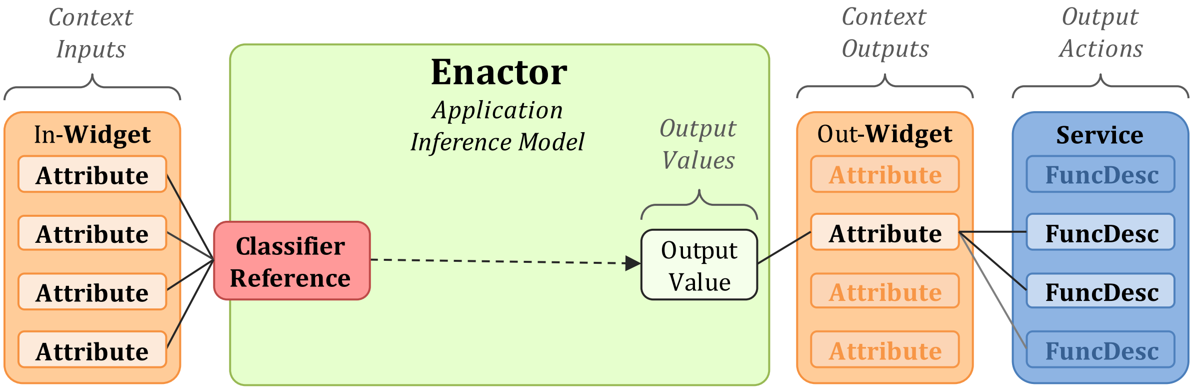


Figure 6.4: Architecture of Enactor framework updated to support classifiers. Only one Reference is used for the classifier; in contrast, for rules one Reference is needed per rule.

The original Enactors supported only rules for inference. However, machine learning classifications are becoming more popular for use in context-aware applications. Therefore, we extended the Enactor framework to support classifiers. We used the Weka toolkit [Hall *et al.*, 2009] for the several classifiers (*e.g.*, J48 decision tree, naïve Bayes, logistic regression, SVM, kNN, AdaBoost) and Jahmm [Francois, 2010] for HMM classifiers.

6.7.3 DECOUPLING FROM THE CONTEXT TOOLKIT

We have developed the Intelligibility Toolkit to be easily decoupled from the Context Toolkit. Should developers choose to use other frameworks to build their applications, they will just need to write an Explainer to generate system-based explanations.

6.8 EXPLANATION EXPRESSION

We describe specific components that make up the data structure for `Explanations`. An Explanation consists of the `Query` passed to the Explainer to generate it and a corresponding `Expression` containing the content of the explanation. Expressions come in two styles: logic expressions for rule traces and multi-arrays for weights of evidence.

6.8.1 RULE TRACE LOGIC EXPRESSIONS

Rule trace logic expressions consist of singular and compound expressions. They are able to represent arbitrary rule expressions, and expressions structured in disjunctive normal form (DNF) or conjunctive normal form (CNF).

6.8.1.1 TERMINAL EXPRESSIONS (CONDITION LITERALS)

We use terminal expressions to represent condition *literals*, which are atomic elements of Boolean expressions. Presently, only nominal and numeric attributes are handled.

PARAMETER

`Parameter` represents a condition literal for a nominal or numeric attribute taking a specific value, *e.g.*, temperature $= 25$°C, room $=$ "*Living Room*". It contains a `name` and a `value`.

COMPARISON AND DUALCOMPARISON

`Comparison` extends `Parameter` to be able to represent inequality (and equality) relations, *e.g.*, temperature ≥ 25°C, frequency ≤ 1500Hz. Comparison adds a `relation` property. We also have `DualComparison` to represent a double-bounded relation, *e.g.*, 1000Hz $<$ frequency ≤ 1500Hz.

NEGATED

`Negated` represents a negated condition literal with the original `Parameter` as its child. It is useful for representing Normal Forms (*e.g.*, negated normal form, DNF, CNF).

6.8.1.2 COMPOUND EXPRESSIONS

We have compound expressions to represent combinations of multiple conditions literals. These are useful for parsing rules into the Expression format and subsequently converting to DNF or CNF.

CONJUNCTION AND REASON

We encode a `Conjunction` as a `List` of `Expressions` connected with the AND (∩) operator. `Reasons` are Conjunctions where each element is a condition literal.

DISJUNCTION AND CLAUSE

We encode a `Disjunction` as a `List` of `Expressions` connected with the OR (∪) operator. `Clauses` are Disjunctions where each element is a condition literal.

NEGATION

`Negation` represents the negation of any `Expression`. Unlike `Negated`, the child expression does not need to be a terminal literal.

DNF

To support standardization for post-processing, we require explanation expressions to be converted into DNF (see Section 6.10.3) as opposed to leaving them in arbitrary structures. A `DNF` is a `Disjunction` of `Reasons`.

CNF

Some explanations may be represented in CNF (*e.g.*, Why Not, see Section 6.10.3.2). A `CNF` is a `Conjunction` of `Clauses`.

6.8.2 WEIGHTS OF EVIDENCE ARRAYS

Weights of Evidence explanations consists of atomic weights that sum together to give a total evidence for inference. We represent this summation of weights with arrays.

6.8.2.1 WEIGHTSEVIDENCE

`WeightsEvidence` represents a one-dimensional weights of evidence. It extends `Reason`, so it is easily amenable to Reducers that also apply to Reasons (see Section 6.11.2.1), or Presenters that are originally designed for Reasons.

6.8.2.2 WEIGHTSEVIDENCEND

In general, weights of evidence may be multiple dimensional (*e.g.*, input features and time for HMM). We use `WeightsEvidenceND` to represent a multi-dimensional array (multi-array) of weights of evidence. Using object-oriented design in `WeightsEvidenceND`, we provide capabilities for arithmetic operations for multi-array weights of evidence (*e.g.*, add, subtract, normalize). These operations are necessary for computing Why and Why Not explanations (see Section 1.1.1). Particularly important for explaining ensemble meta-classifiers, new dimensions can be added to `WeightsEvidenceND` using the `collate` function (*e.g.*, see Section 6.10.8).

`WeightsEvidenceND` can be reduced to `WeightsEvidence` using `DimensionReducer` (see Section 6.11.2).

6.9 QUERY

We describe three types of queries currently implemented in the Intelligibility Toolkit to support various question types.

6.9.1 BASE QUERY

The base `Query` provides an encapsulation for posing a question to an explainer by specifying a question type, the context to ask, and about which time (to support asking about a historical event). For example, to ask why availability was inferred as it was at 4pm, we create the query (in pseudo-code):

```
new Query("availability", Query.WHY_QUESTION, "4pm")
```

6.9.2 ALTQUERY

`AltQuery` extends `Query` and includes an alternative target output value to facilitate explanation types Why Not, and How To.

6.9.3 INPUTSQUERY

`InputsQuery` extends `Query` and allows the setting of input values; this supports What If explanations.

6.9.4 EXTENDING QUERY

Basic Query	AltQuery	InputsQuery
What Certainty When Why Inputs Outputs Description Situation	Why Not How To	What If

Table 6.1. Summary of currently supported question types grouped by the Query class that supports them.

While the Queries already supports a wide range of question types, developers may want to provide explanations for questions that we had not explored, or more esoteric explanation types relevant to their specific applications. It is easy to support a question type by specifying a new name for the `question` parameter in the Query. Unfortunately, none of the existing Explainers will recognize and be able to answer the new question. This will require new code and algorithms. We discuss how a toolkit programmer may allow Explainers to support new question types in Section 6.10.

6.10 EXPLAINER ALGORITHMS

We describe the algorithms to generate various explanation from five inference models, and defer a more mathematically rigorous derivation in Appendix B. For system-based explanation types, we focus on explanations from Enactors, and describe base explanations for the WEKA toolkit in Appendix B.3. For model-based explanation types, we describe algorithms for explaining Rules, Decision Trees, naïve Bayes, and hidden Markov models, and Bagging. We defer descriptions of algorithms for explaining other machine learning classifiers in Appendix B.

6.10.1 ENACTOR BASE EXPLAINER

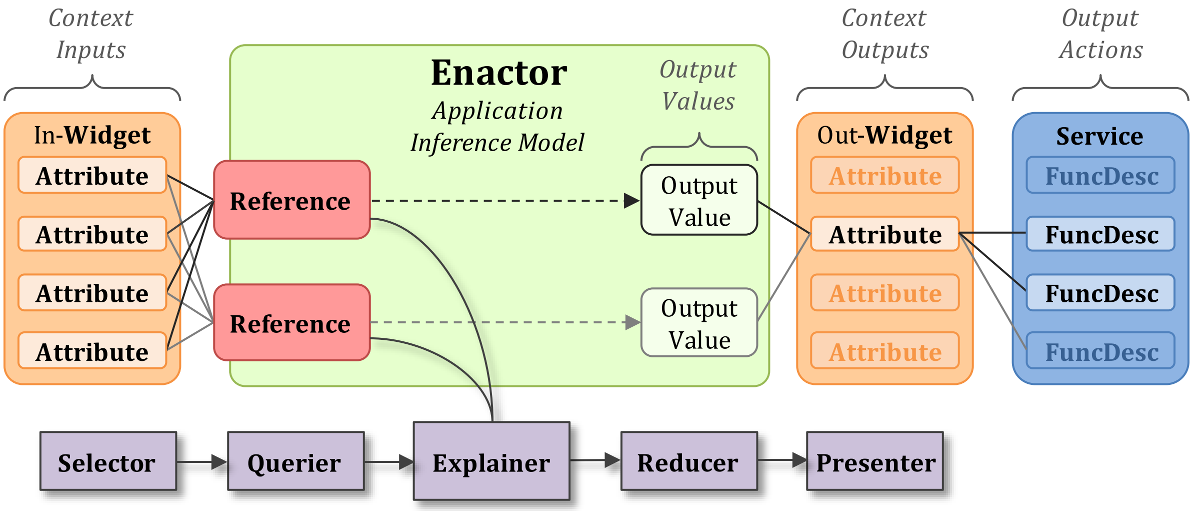


Figure 6.5: Architecture of components of the Intelligibility Toolkit integrated into the Enactor framework for a typical context-aware application built with the Context Toolkit.

We describe how system-based explanations are generated from Enactors in the updated Enactors framework (Section 6.7.1). Figure 6.5 illustrates the relation of the components of the Intelligibility Toolkit with the Enactor framework.

6.10.1.1 WHAT EXPLANATION

This simply returns the Output value of the `EnactorReference` that is triggered.

6.10.1.2 WHEN EXPLANATION

This retrieves the timestamp of when the Widget `Attribute` value was changed.

6.10.1.3 INPUTS EXPLANATION

The Inputs explanation reports the name and value of each context (`Widget` attribute) used as input in the Enactor model, *i.e.*, from all References.

6.10.1.4 OUTPUTS EXPLANATION

The Outputs explanation reports a `List` of output values that the Out-Widget of the Enactor may take.

6.10.1.5 WHAT IF EXPLANATION

A What If explanation sets input context values set by the user (through `InputsQuery`), and tests it on all References. It reports the output value associated with the Reference that gets triggered. Model-specific explanations are generated from different Explainers for each model.

6.10.1.6 DESCRIPTION EXPLANATION

We use the `DescriptionExplanationDelegate` (see Section 6.10.9) to maintain a map of textual information for contexts used in the inference model.

6.10.2 MODEL-BASED EXPLAINERS

The base Explainer leaves model-based explanations unimplemented and defer to concrete explainers to handle. In the next sections, we describe several Explainers and the algorithms to generate explanations from various inference models. First, we cover Rule Trace explainers, then Weights of Evidence explainers.

6.10.3 Disjunctive Normal Form (DNF) Trace Explainer

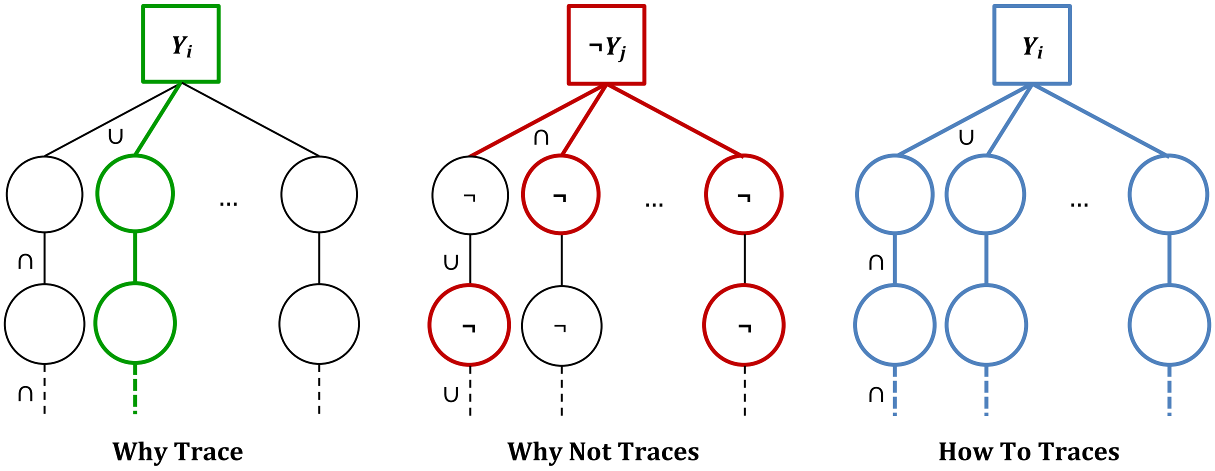


Figure 6.6: Diagram representation of extracting Why explanation(s) and extracting Why Not explanations from a system of DNF trees. (Left) Explains why *i*th class was inferred, selecting conjunction traces that are satisfied by the current input values. (Middle) Negating the DNF tree for rules inferring the *j*th class produces a tree in conjunctive normal form (CNF) after applying De Morgan's Laws. Note that condition literals are negated. All disjunction traces in the CNF negated tree are satisfied, highlighting specific negated condition literals that are satisfied by the current input values. (Right) Explains how to infer the *i*th class by returning the full DNF tree that can infer that class.

We use the disjunctive normal form (DNF) as the core data structure from which to generate rule trace explanations. We assume that for sufficiently simple context-aware applications, rules can be converted into DNF (albeit not necessarily efficiently) and so can decision trees. We store the rules as a system of DNF trees, one for each outcome value. Once we have the DNF trees, we can generate Why, Why Not and How To explanations (see Figure 6.6). Detailed proofs are in Section B.5.

6.10.3.1 WHY EXPLANATION

This explanation selects the rule(s) that was satisfied to produce the actual output (see Figure 6.6, Left).

6.10.3.2 WHY NOT EXPLANATION

This explanation selects rules that would achieve the target output value and, for each trace, identifies unsatisfied conditionals and returns a disjunction of traces containing these conditionals (Figure 6.6, Middle).

6.10.3.3 HOW TO EXPLANATION

This explanation returns the full DNF tree of the rule that achieves the target output, where each trace is a rule of how to achieve the target output value (Figure 6.6, Right).

6.10.4 RULES EXPLAINER

We implement an explainer for rules in the Enactor framework that provides a Rule Trace style of explanations.

The Enactor framework supports one rule per Reference and we enforce one Reference per output state. A decision model with multiple output states will have multiple rules, one per state. We make rules explainable by converting them into disjunctive normal form (DNF) by recursively applying De Morgan's Law, double negative elimination, and the distributive law (see Figure B.2 and Figure B.3 for a simple algorithm for the conversion). To illustrate how each explanation type is generated, we shall use the set up described in Table 6.2 and Figure 6.7.

6.10.4.1 WHY, WHY NOT, AND HOW TO EXPLANATION

These explanations are generated from the converted DNF trees from Section 6.10.3.

6.10.4.2 CERTAINTY EXPLANATION

Enactor rules currently do not compute uncertainty, though uncertainty in inputs can be propagated descriptively.

6.10.4.3 DECISION TREE RULE TRACE EXPLAINER

We implement an explainer for decision trees that provides a Rule Trace style of explanations.

This explainer generates explanations from the Weka [Mark *et al.*, 2009] J48 implementation of the C4.5 decision tree [Quinlan, 1993]. There are some differences between decision trees and Enactor rules. Decisions for inferring the output are made from the top down, instead of from the bottom up as for rules; and decision trees encode traces for multiple output values within a single tree, unlike a rule-based structure. So DNF rules are created by traversing all paths in the tree and grouping paths by output values at their leaves (see Figure B.5 and Figure B.6 for the parsing algorithms). In DNF, we can generate explanations in the same way as for rules. To illustrate how each explanation type is generated, we shall use the set up described in Table 6.2 and Figure 6.8.

6.10.4.4 WHY, WHY NOT, AND HOW TO EXPLANATION

These explanations are generated from the converted DNF trees from Section 6.10.3.

6.10.4.5 CERTAINTY EXPLANATION

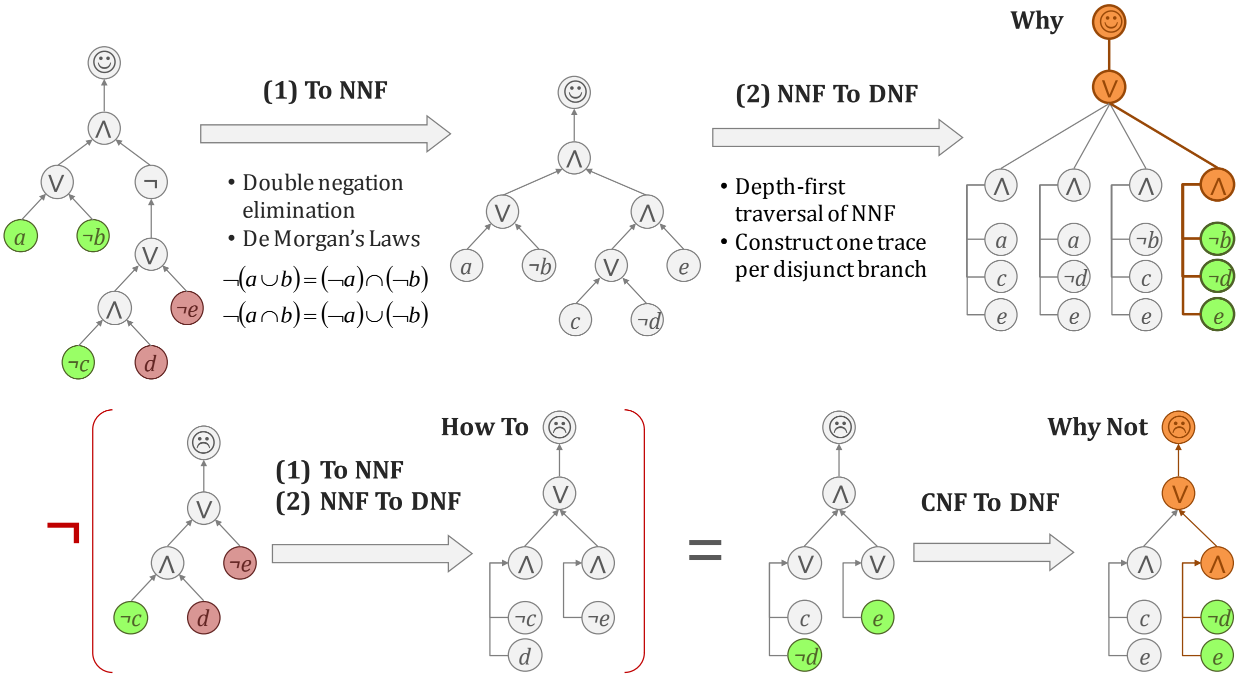
Decision trees are built from statistical data, so they can model certainty from the probability distribution of remaining data points at each leaf.

Input Conditionals		Output Values (Availability)	
a:	Location = Home	☺ =	Available
b:	Sound = In a Conversation	😐 =	Somewhat Unavailable
c:	Time = Evening	☹ =	Unavailable
d:	Schedule = Meeting		
e:	Contacter = Family/Friend		

Table 6.2. Pedagogical example of input conditionals and output values for rule and decision tree. This describes an application to infer a user's availability based on his Location, Sound activity around him, the Time of the day, his Schedule, and who is Contacting him.

a b c d e ≡ a $\neg b$ $\neg c$ $\neg d$ e

Input state (a, ¬b, ¬c, ¬d, e): the user is at Home, not in a Conversation, not in a Meeting, and is being contacted by a Family member or Friend.

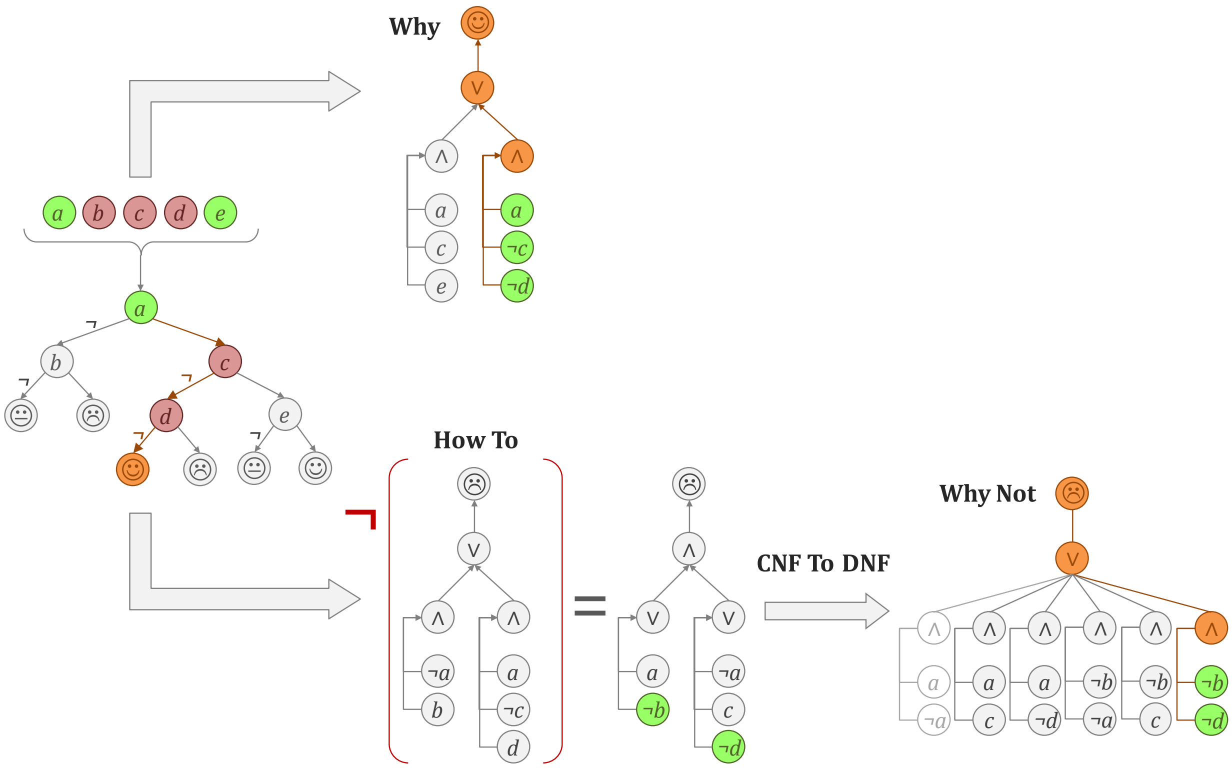

(1) To NNF
• Double negation elimination
• De Morgan's Laws
$\neg(a \cup b) = (\neg a) \cap (\neg b)$
$\neg(a \cap b) = (\neg a) \cup (\neg b)$
(2) NNF To DNF
• Depth-first traversal of NNF
• Construct one trace per disjunct branch
Why
¬
How To
(1) To NNF
(2) NNF To DNF
=
Why Not
CNF To DNF
a
¬b
¬c
d
¬e
c
¬d
e

Why *available* (☺)? Because the user is not in a Conversation (¬*b*), is not in a Meeting (¬*d*), and being contacted by a Family/Friend (*e*).

Why Not *unavailable* (☹)? Because he is *not* in a Meeting (¬*d*) and is being contacted by a Family/Friend (*e*).

How To infer *unavailable* (☹)? It needs to be before Evening time (¬*c*) and he needs to be in a Meeting (*d*); or he is contacted by a Coworker (not Family/Friend; ¬*e*).

Figure 6.7: Conversion of a set of rule tree expressions into separate trees in disjunctive normal form (DNF) to generate explanations from Rules.



Why *available* (☺)?	Because the user is at Home (a), it is *not* Evening time ($\neg c$), and he is *not* in a Meeting ($\neg d$).
Why Not *unavailable* (☹)?	Because he is *not* in a Conversation ($\neg b$) and he is *not* in a Meeting ($\neg d$).
How To infer *unavailable* (☹)?	He needs to be *not* at Home ($\neg a$) and in a Conversation (b); or at Home (a), not during the Evening ($\neg c$), and be in a Meeting (d).

Figure 6.8: Conversion of a decision tree into separate trees in disjunctive normal form (DNF) to generate explanations from Decision Trees.

6.10.5 WEIGHT OF EVIDENCE EXPLAINER

This explainer forms the abstract basis of our subsequent weights of evidence explainers. It works on the basis that an inference is due to a total evidence, where this evidence may support or oppose the inference, and can be due to a *sum* of underlying *atomic* weights of evidence. These weights may be due to the input feature values voting for or against an inference. Depending on the inference model, there may also be more *dimensions* of atomic weights of evidence. For simplicity, we denote an atomic weight as f_{ir}, and the space of all atomic weights as R. A total evidence can thus be represented as the sum:

$$g_i = \sum\nolimits_{r \in R} f_{ir} \tag{6.1}$$

Equation (B.35) requires that the explainer is able to derive a *linear additive* expression of atomic units. This is easy for linear classifiers (*e.g.*, linear SVM), but in general, isotonic (monotonic increasing) transformations may be required (*e.g.*, see explainer for naïve Bayes in Section B.15). We defer these steps to later sections describing the concrete explainers.

We shall next show how with these *absolute* weights of evidence, we can derive weights of evidence explanations for Why and Why Not questions.

6.10.5.1 ABSOLUTE EVIDENCE

This explains the confidence of inference, p_i, as a total evidence due to the sum of atomic weights

$$g_i = \sum\nolimits_{r} f_{ir} \tag{6.2}$$

where f_{ir} is the rth atomic weight of evidence.

6.10.5.2 WHY NOT EXPLANATION

This explains why the jth class was *not* inferred over the ith class. In other words, why the ith class was inferred instead of the jth class, *i.e.*, $p_i \geq p_j$:

$$\Delta g_{ij} = g_i - g_j = \sum\nolimits_{r} \Delta f_{ijr} \geq 0 \tag{6.3}$$

where $\Delta f_{ijr} = f_{ir} - f_{jr}$ and we assume that the atomic weights of evidence are separable by each atomic unit. Note that this is different from the Why Not explanation of a *rule trace* that explains why the jth class was not inferred. In this case, the jth class may have been inferred, but just not with the highest certainty among all class values.

6.10.5.3 WHY EXPLANATION

This explains why the ith class was inferred over all m other class values j, *i.e.*, $p_i \geq p_j, \forall j$. Consequently, Equation (6.3) holds for $\forall j$, such that we can sum over $\forall j$ to get the Why explanation:

$$\Delta g_{i\forall} = \sum_{j=1}^{m} \Delta g_{ij} = \sum_{j=1}^{m} \sum_{r} \Delta f_{ijr} \geq 0 \tag{6.4}$$

6.10.5.4 CERTAINTY EXPLANATION

Assuming that at the system level, the model will produce a distribution of certainty for inferring each class value, the Certainty explanation returns this distribution:

$$p = \sum_{j=1}^{m} p_j \tag{6.5}$$

where p is the total certainty (usually normalized as a probability to 1) and p_j is the certainty for inferring the jth class value.

6.10.5.5 HOW TO EXPLANATION

This explanation depends on underlying model and we defer this to the later sections.

6.10.6 NAÏVE BAYES EXPLAINER

We implement an explainer for naïve Bayes that provides a Weights of Evidence style of explanation. The naïve Bayes classifier is a simple classifier that uses Bayes theorem to classify values. It assumes that input features are conditionally independent of one another. Our explanation for naïve Bayes inference is an extension of [Poulin *et al*. 2006] for multi-class problems.

The posterior probability that the ith class is inferred ($y = y_i$) from a set of m class values given the observed instance input feature values x:

$$P(y = y_i|x) = P(y_i|x_1, x_2, \cdots, x_n) \cong P(y_i) \prod_{r=1}^{n} P(x_r|y_i) \tag{6.6}$$

where x_r is an input feature of n possible values. The probability is calculated from the prior probability that a class would be in, $P(y_i)$, and the conditional probabilities of each feature value given the class, $P(x_r|y_i)$. For notation convenience, let us define

$$p_{ir} = \begin{cases} P(y_i) & r = 0 \\ P(x_r|y_i) & r > 0 \end{cases} \qquad p_i = P(y_i|x)$$

Then Equation (6.6) becomes

$$p_i \cong \prod_{r=0}^{n} p_{ir} \tag{6.7}$$

Taking a log transform of Equation (6.7) gives the linear expression for the weights of evidence explanation:

$$\begin{gathered} \log(p_i) = \sum_{r=0}^{n} \log(p_{ir}) \\ g_i = \sum_{r=0}^{n} f_{ir} \end{gathered} \tag{6.8}$$

where $f_r = \log(p_{ir})$.

With Equation (6.8), naïve Bayes can be explained as the sum of evidence

1. Prior probabilities of *selected class value*, $\log(p_{ir})\,, r = 0$
2. Due to each *feature value*, $\log(p_{ir})\,, r > 0$

and we can derive a Why and Why Not explanation by substituting Equation (6.8) into Equation and Equation , respectively.

6.10.6.1 HOW TO EXPLANATION

This shows weights of all features due to *normalized* values of the features, so the user can make sense of the general impact of each feature. (i) For nominal features, they only take a value of 0 or 1, so their evidence will either be 0 or f_{ir}. (ii) For numeric features, they are commonly modeled by the Normal distribution; we "normalize" numeric features to a value of one standard deviation, σ_{ir}, from the feature mean given the class value: $\mu_{ir} \pm \sigma_{ir}$.

In terms of what the user can do with input values to get a desired target output, if all values are nominal, we can permute all combinations and return those that achieve the target output. If multiple inputs are numeric, this becomes intractable. Instead, we could use a How To If explanation.

6.10.6.2 HOW TO IF EXPLANATION

This provides a tractable form of How To explanations (for nominal and numeric features) by constraining all feature values except one. For a numeric feature that is not fixed, we can vary the input value as it deviates *from the mean* and determine the threshold at which the outcome is achieved (or fails, if it is the opposite relation). A generalization of this with increased interactivity is the How To + What If explanation.

6.10.6.3 HOW TO + WHAT IF EXPLANATION

One way to help users appreciate the influence of each weight is to allow users to speculate on the outcome with selected feature values. The What If explanation supports this, but does not necessarily start with sensible values to help users learn. The How To + What If explanation starts with the target ith class value, and provides a set of *mean* feature values that satisfies this output, μ_{ir}. The user can then tweak the feature values to see if the target output value would still be inferred.

6.10.7 HMMEXPLAINER

We implement an explainer for HMMs that provides a Weights of Evidence style of explanation.

A hidden Markov model (HMM) is a Bayesian network that models the probability of a *sequence* of hidden states given a sequence of observations (input features with respect to time). First-order Markov models assume that only the previous state affects the next, and only the current state influences the current observation. For detailed information on HMMs, please refer to [Rabiner, 1989]. We can derive a weights of evidence explanation for HMMs in a similar manner as we did for naïve Bayes.

We consider HMMs with observation vectors for each time step, specifically with n input features. At time step t, the observation is $\vec{x}_t = (x_{t1}, x_{t2}, \dots, x_{tn})$.

The probability of inferring state sequence s with length T, given the observation sequence x is:

$$\begin{aligned} P(s|x) &= P(s_1)\left(\prod_{t=2}^{T} P(s_t|s_{t-1})\right)\left(\prod_{t=1}^{T} P(\vec{x}_t|s_t)\right) \\ &= \pi(s_1)\left(\prod_{t=2}^{T} A(s_t|s_{t-1})\right)\left(\prod_{t=1}^{T} B(\vec{x}_t|s_t)\right) \end{aligned} \tag{6.9}$$

where $P(s_1) = \pi(s_1)$ is the *prior* probability that a state is s_1, $P(s_t|s_{t-1}) = A(s_t|s_{t-1})$ the *transition* probabilities from state s_{t-1} to s_t, and $P(\vec{x}_t|s_t) = B(\vec{x}_t|s_t)$ the *emission* probabilities of the observations given the state sequence.

To allow features to individually provide evidence, we need to make a *naive assumption* (similar to what is done for naive Bayes) that the features are independent of one another given any state, *i.e.*,

$$P(\vec{x}_t|s_t) = P(x_{t1}, x_{t2}, \dots, x_{tn}|x_t^i) \cong \prod_{r=1}^{n} P(x_{tr}|s_t) \tag{6.10}$$

We define a new parameter for the HMM, $\tilde{B}(x_{tr}|s_t) = P(x_{tr}|s_t)$, which is a naive emissions probability matrix representing the probability of observing input value x_{tr} given hidden state s_t.

Substituting Equation (B.97) into Equation (B.96) gives

$$P(s|x) = \pi(s_1)\left(\prod_{t=2}^{T} A(s_t|s_{t-1})\right)\left(\prod_{r=1}^{n} \tilde{B}(x_{tr}|s_t)\right) \tag{6.11}$$

For notation convenience, we rewrite Equation (B.98) as:

$$p_s \cong \prod_{t=1}^{T} \prod_{r=0}^{n} p_{str} \tag{6.12}$$

where

$$p_{str} = \begin{cases} \pi(s_1) & \text{, if } r = 0, t = 1 \\ A(s_t|s_{t-1}) & \text{, if } r = 0, t > 1 \\ \tilde{B}(x_{tr}|s_t) & \text{, if } r > 0 \end{cases}$$

Taking a log transform on Equation (B.96) gives the weights of evidence explanation:

$$g_s = \sum\nolimits_{t=1}^{T} \sum\nolimits_{r=0}^{n} f_{str} \tag{6.13}$$

where

$$f_{str} = \log(p_{str}) = \begin{cases} \log(\pi(s_1)) & \text{, if } r = 0, t = 1 \\ \log(A(s_t|s_{t-1})) & \text{, if } r = 0, t > 1 \\ \log\left(\tilde{B}(x_{tr}|s_t)\right) & \text{, if } r > 0 \end{cases}$$

So, with the naive assumption of independence among features, HMMs can be explained as the sum of evidence of:

1. Prior probabilities of selected state, $(r = 0, t = 1)$
2. Weights of Evidence due to each state transition, $(r = 0, t > 1)$
3. Weights of Evidence due to feature value at time sequence step, $(r > 0)$

6.10.7.1 INPUTS AND WHAT IF EXPLANATION

Though not formally an input, these explanations should also include state transitions. For the What If explanation, the user may want to speculate if a previous state was different, even though hidden states are actually inferred.

6.10.7.2 WHY, WHY NOT, HOW TO, AND CERTAINTY EXPLANATIONS

Similar to the explainer for naïve Bayes, but with added evidence for time.

6.10.7.3 REDUCING DIMENSIONALITY

If an application has many input features and/or a long sequence (*i.e,.* large n and/or large T), there may be too many weights of evidence to show to end-users. To reduce this dimensionality and make this more interpretable, we can sum evidence *by feature* across time (see Figure 6.12), or present evidence *by time* and sum evidence across features for each observation. Formally, this is done using DimensionReducer (see Section 6.11.2).

6.10.8 BOOTSTRAP AGGREGATION (BAGGING) EXPLAINER

Ensemble meta-classifiers are popular techniques to further improve the accuracy of classifiers. Two popular ensemble methods are Bootstrap Aggregation and AdaBoost. We describe how to generate explanations for Bagging in this section, and for AdaBoost in Appendix B.22.

Bootstrap aggregation (also called Bagging) [Breiman, 1996] takes a training dataset, D, and creates C new training sets by sampling examples from D with replacement. This method is called boostrap sampling. It then trains C versions of the base classifier, one classifier for each new training set. Overall inference is done by voting the inference of each of the base classifiers:

$$p_i = \frac{1}{Z}\sum_{c=1}^{C} p_{ic} \tag{6.14}$$

where p_{ic} is the probability of the cth classifier inferring the ith class, and Z is a normalization constant.

Using Equation (B.126), we can derive a weights of evidence explanation for inferring the ith class in terms of classifiers:

$$g_i = \sum_{c=1}^{C} p_{ic} \frac{g_{ic}}{\text{value}(g_{ic})} \tag{6.15}$$

where g_{ic} is the linearly separable weights of evidence expression for the cth base classifier and $\text{value}(g_{ic}) = \text{sgn}(g_{ic})|g_{ic}|$ is the value of the total weights of evidence. The term $\frac{g_{ic}}{\text{value}(g_{ic})}$, which has value $= 1$ or -1, allows us to normalize the magnitude of g_{ic} such that we can retain the relative weights of evidence due to each base classifier g_{ic} and scale each of them to the certainty of that classifier, p_{ic}.

Note that because this meta-classifier explainer requires weights of evidence explanations from the base classifier, it will only work if the classifier has an Explainer that can generate weights of evidence. To give an idea of what the atomic weights of evidence represent for several bagged classifiers, we work out the expression for bagged naïve Bayes, though they are automatically generated using Equation (6.10).

6.10.8.1 RANDOM (BAGGED) NAÏVE BAYES

Adapting the weights of evidence explanation for naïve Bayes within an ensemble classification, Equation (6.8) becomes:

$$g_{ic} = \log(p_{ic}) = \sum_{r=0}^{n} \log(p_{icr}) \tag{6.16}$$

Substituting Equation (6.11) into Equation (6.10), we get

$$g_i = \sum_{c=1}^{C} p_{ic} \frac{g_{ic}}{\text{value}(g_{ic})} = \sum_{c=1}^{C} \sum_{r=0}^{n} f_{icr} \tag{6.17}$$

where

$$f_{icr} = \frac{p_{ic}}{\text{value}\big(\log(p_{ic})\big)} \log(p_{icr})$$

Hence the weights of evidence for a bagged naïve Bayes meta-classifier consists of two dimensions: input features and classifier.

6.10.9 DESCRIPTIVEEXPLAINERDELEGATE

We use a delegate to manage textual descriptions to provide Description explanations. It uses a system of hash maps to store various types of descriptions for each context (input or output). Using insight gained from designing Laksa, a high-fidelity intelligible application prototype (see Sections 7.2 and 9.3), we developed the `DescriptiveExplainerDelegate` to include maps for textual content such as:

- Descriptive text explaining **terminology** or describing concepts
- **Pretty names** to use in place of original technical terms for contexts or their nominal (discrete) values
- **Units** of the context, *e.g.*, seconds for time, Hz for frequency, Watts for power (energy/time)
- Longer text to provide the **design rationale** or **justification** for the context or the implication of the context taking various values.

In general, this can be extended to include other types of textual descriptions that developers may be interested to include in their applications.

6.10.10 OTHER IMPLEMENTED EXPLAINERS

Rule Trace	Weights of Evidence
Rules	Linear Regression
Decision Trees	Logistic Regression
	Naïve Bayes
	HMM
	Decision Trees *
	Random Forest
	Linear SVM
	kNN
	AdaBoost
	Bagging

Table 6.3. Summary of inference models supported by the Intelligibility Toolkit grouped by style of explanation each explainer generates. * The toolkit provides Rule Trace *and* Weights of Evidence explainers for decision trees, *e.g.*, J48 has J48RuleTraceExplainer and J48EvidenceExplainer.

The Intelligibility Toolkit currently supports at least 11 types of explainers for popular inference models in context-aware applications and machine learning applications (see Table 6.3). Algorithms for generating explanations from these models are in Appendix B.

6.10.11 EXTENDING EXPLAINERS

System-based	Model-based	Application-based
What	Certainty	Description
When	Why	Situation
Inputs	Why Not	
Outputs	How To	
What If		

Table 6.4. Summary of explanation types grouped by dependency. Toolkit programmers only need to implement Model and Application-dependent explanation types.

New explainers can be developed to (i) explain new inference models, or (ii) support new explanation types, which are currently unsupported in the Intelligibility Toolkit. The latter goal also includes providing new styles of explanations for explanation types that are currently supported. Here, we focus on how developers may write new explainers to explain new inference models.

Given the 12 explanation types we seek to support (see Section 6.3 and summary in Table 6.4), it can be cumbersome for each new explainer to implement algorithms for all of them. Fortunately, there are common dependencies that developers may take advantage of. Six system-dependent explanation types are automatically supported if developers build applications using the Enactors framework [Dey and Newberger, 2009]. Otherwise, these explanation types can also be implemented for different frameworks (*e.g.*, for Android mobile programming).

Implementing the four model-dependent explanation types is also simplified with the `DnfTraceExplainer` and `WeightsOfEvidenceExplainer`, which implement these explanation types. To provide explanations for a rule-based system similar to Enactor rules or for a decision tree model, the inference model need only be parsed into DNF trees, one DNF tree per outcome value. Explanations can then be automatically generated for Why, Why Not and How To using `DnfTraceExplainer` as a delegate. To provide explanations in terms of weights of evidence, the explainer need only implement the sum of atomic weights of evidence, g_i (see Equation (6.2)), or the sum of relative weights of evidence, Δg_{ij} (see Equation (6.3)). Explanations can then be automatically generated for Certainty, Why, Why Not and How To using `WeightsOfEvidenceExplainer`.

Finally, to support application-dependent explanation types, such as Description and Situation, explainers will need to be hand-crafted to suit the application being built. Description explanations require custom text content and Situation explanations may involve custom information and representations.

Note that, though recommended, it is not necessary for explainers to implement all 12 explanation types in Table 6.4.

6.11 REDUCER

Generated explanations may contain too much information and be overwhelming to end-users. In this section, we describe the Reducer component to simplify the explanation data structures before passing them on to Presenters. The Intelligibility Toolkit currently supports two types of reducers: Logic Reducers for rule traces, and the Dimension Reducer for weights of evidence.

6.11.1 LOGIC REDUCERS

Rule trace explanations can be overwhelming in two ways: (i) too many reasons (*e.g.*, numerous ways to achieve a target output value), and (ii) each reason being too long (*e.g.*, numerous inputs with required values to cause the output value). The latter case can happen when many sensors and feature values are used to build the models, as is the case for accurate learned systems. The Intelligibility Toolkit provides Disjunction Reducers to reduce the number of reasons via various schemes, Conjunction Reducers to reduce the length of each rule trace reason, along with Reducers that can handle any logical expression (*e.g.*, QmcReducer that uses the Quine-McCluskey algorithm to minimize logic expressions).

6.11.1.1 DISJUNCTION REDUCERS

We describe several `DisjunctionReducers` to reduce the number of `Reason` traces.

FIRSTDREDUCER

This reducer simply selects the first reason in the DNF structure of the explanation.

SHORTESTDREDUCER

This reducer selects the reason with the shortest length as the explanation.

6.11.1.2 CONJUNCTION REDUCERS

We describe several `ConjunctionReducers` to shorten the length of each `Reason` trace.

TRUNCATIONCREDUCER

This reducer simply truncates the `Reason` trace to a specified length.

FILTERCREDUCER

This reducer retains only the condition literals (`Parameter`) in the `Reason` that describe input features specified in a list, dropping others. Input features can be selected for their saliency, easy of understanding, or even for privacy sensitivity, depending on the application.

6.11.1.3 MINIMIZATION REDUCERS

In order to simplify the explanations, the previous Reducers unfortunately lose information by discarding them. An alternative method to simplify explanations is to minimize the Boolean expression of the explanation. This discards only the redundant information and may both reduce the number of reason traces and the length of each reason trace. Several methods exist for logic minimization such as Karnaugh mapping, the Quine-McCluskey algorithm, the Espresso heuristic minimization program. We currently support the Quine-McCluskey algorithm with the `QmcReducer`.

NUMERIC CONJUNCTION REDUCER

This reducer selects condition literals in a reason and merges inequalities that relate to the same numeric feature. For example,

$$\begin{aligned}(x_r > 5) \cap (x_r > 3) &\equiv \quad x_r > 3 \\ (x_r \leq 5) \cap (x_r > 3) &\equiv 3 < x_r \leq 5\end{aligned}$$

QUINE-MCCLUSKEY REDUCER

This reducer, `QmcReducer`, uses the Quine-McCluskey algorithm[2] to minimize a Boolean expression by removing redundant condition literals. It can be applied to the rule structure in any Boolean expression. We ported the implementation of the Quine-McCluskey algorithm in the Truth Table Solver [Ahmed, 2011] to work with the `Expression` data structure.

6.11.1.4 COMPOUNDREDUCER

We provide `CompoundReducer` to provide a convenient means to apply multiple reducers sequentially on an Explanation.

6.11.1.5 EXTENDING LOGIC REDUCER

New logic reducers can be implemented to support other heuristics to simplify explanations. Simple reducers, which assume that explanations are structured in DNF, can focus on reducing the number of reasons by extending `DisjunctionReducer`, or focus on reducing the length of each reason by extending `ConjunctionReducer`. More sophisticated reducers can be developed for reducing explanations by directly extending the base `Reducer`. In particular, more efficient algorithms can be implemented for logic minimization.

Reducers may be used in alternative manners that do not necessarily reduce the length of explanations. For example, Metaxas [2010] suggested that users interpret explanations more

[2] Quine-McCluskey algorithm. http://en.wikipedia.org/wiki/Quine-McCluskey_algorithm Retrieved 1st March, 2012.

quickly when terms have higher *affinity* (*e.g.*, “driving” and “running” have higher affinity than “running” and “talking”). Therefore, we may implement an `AffinityReducer` that applies to `Conjunctions` that re-arranges logical terms (`Parameters`) to group them by affinity, instead of reducing their lengths.

6.11.2 DIMENSION REDUCER

Simple weights of evidence explainers can generate weights in terms of just the input features. However, there may be multiple dimensions for the weights of evidence (*e.g.*, the explainer for hidden Markov Models (HMMs) has a dimension for input features and time), leading to an issue of dimensionality explosion, where the number of weights of evidence equals the product the lengths of each dimension. Once again this can overload end-users with too much information. One way to reduce this complexity is to aggregate the weights of evidence along some dimensions. For example, for HMM explanations, we may aggregate along time to sum all weights that are regarding the same time step, or we may aggregate along input features to sum all weights that are regarding the same input feature. The Intelligibility Toolkit provides this functionality with `DimensionReducer`, which reduces a multi-dimensional weights of evidence array, `WeightsEvidenceND`, to fewer dimensions chosen by the developer.

A typical use of `DimensionReducer` may be to reduce a multi-dimensional weights of evidence, `WeightsEvidenceND`, to just the one dimension, `WeightsEvidence`, specifically the input features dimension. This then resembles a single rule trace and we can apply `ConjunctionReducers` to further simplify the explanations if needed.

6.11.2.1 CONJUNCTION REDUCERS FOR WEIGHTS OF EVIDENCE

A typical use of `DimensionReducer` may be to reduce a multi-dimensional weights of evidence, `WeightsEvidenceND`, to just the one dimension, `WeightsEvidence`, specifically the input features dimension. This then resembles a single rule trace and we can apply some `ConjunctionReducers` to further simplify the explanations if needed.

REMAINDERCREDUCER

This reducer truncates the `Reason`, but also accumulates remaining weights of evidence that were removed into a remainder term.

NEGLECTCREDUCER

This reducer removes each term from the weights of evidence if its absolute of its numeric value, f_r, is below a threshold, ϵ, *i.e.*, $|f_r| < \epsilon$. It is useful to neglect weights that are approximately zero.

6.12 PRESENTER

Along with `Querier` (see Section 6.13), the `Presenter` component provides a separation of the presentation of explanations from their content information to follow the principle of Model-View-Controller (MVC) separation. Presenters need to implement the method `render(Explanation)`to receive the generated Explanation to present. We describe several Presenters for different platforms.

6.12.1 STRINGPRESENTER

`StringPresenter` is a basic presenter that just renders explanations as a String text. Together with `DescriptionExplainerDelegate`, this presenter can print the explanation with pretty names and units.

6.12.2 DESKTOP-BASED PRESENTERS (JAVA SWING)

We describe several Presenters implemented in Java Swing to build intelligible context-aware applications with desktop-based GUIs.

TABLEPANELPRESENTER

`TablePanelPresenter` renders explanations in a table showing one `Reason` at a time. Each row renders a `Parameter` (conditional literal) with the `name` in the first column and `value` in the second column. See Figure 6.9 for an example of this presenter.

TYPEPANELPRESENTER

`TypePanelPresenter` extends `TablePanelPresenter` to render the explanation differently depending on the explanation type. For example, Why and Why Not explanations render with a bar chart with multiple rows, Inputs explanations render with just the numeric values for each input feature, and What and Certainty explanations render with one value. See Figure 6.9 for an implementation of this presenter.

6.12.3 MOBILE-BASED PRESENTERS (ANDROID)

As part of the implementation for Laκsa2, an intelligible mobile application prototype (see Figure 9.1), we have developed several Presenters on the Android phone to render explanations of various context types and inference models: `AvailabilityPresenter`, `PlacePresenter`, `SoundPresenter`, `MotionPresenter`.

6.12.4 BUILDING YOUR OWN PRESENTERS

To support UI code of any framework, the base `Presenter` is a Java *interface*. While Presenters provide convenient functionality to present explanations with only "glue" code, they are limited in their design, and platform (*e.g.*, only a few Java Swing and Android Presenters are currently implemented). Furthermore, each application will likely require customized presentation and UI code. For example, a map-based application may use a map UI to explain Place inference, while a motion recognition application may draw physical diagrams to explain how accelerometer features influenced the inference.

The main requirements for Presenters are to implement the interface method `render(Explanation)`, and update the UI correspondingly. It is the responsibility of the Presenter to parse the Explanation and understand how to interpret it. The Presenter may also render explanations differently depending on the type of Query it is in response to. Presenters are also tightly coupled to the source Explainer (pertaining to interpreting the generated Explanation structure), and the individual input contextual factors (pertaining to domain information). However, it is not necessary for toolkit programmers of Presenters to know how to generate the explanations (that is for programmers of Explainers).

6.13 QUERIER

`Queriers` provide the control interface for the Model-View-Controller (MVC) architecture for intelligibility. They facilitate end-users to ask questions types and generate `Queries` to be passed to `Explainers`. Similarly to Presenters, they depend on the application platform.

6.13.1 DESKTOP-BASED QUERIERS (JAVA SWING)

We describe a Querier implemented in Java Swing to build intelligible context-aware applications with desktop-based GUIs.

QUERYPANEL

In our demo applications (see Section 6.15), we implemented `QueryPanel` as a GUI element for the Querier to allow users to ask questions for intelligibility. It consists of a drop-down menu (`JComboBox`) to select the question to ask. This facilitates creating a base `Query` after the user selects the question (*e.g.*, see Figure 6.9). To facilitate creating an `AltQuery` for asking Why Not and How To questions, it displays a second drop-down menu showing the possible Output values (*e.g.*, see Figure 6.11, Right). To facilitate creating an `InputQuery` for a What If question type, it also provides `WhatIfPanel`, which allows end-users to choose values of Inputs (*e.g.*, see Figure C., Right).

6.13.2 TEXT-BASED QUERIERS

Queriers need not be GUI-based to allow users to ask questions. Here, we describe a text-based Querier.

QUERYPARSER

As an alternative to GUI-based interaction, `QueryParser` parses text input to interpret what question the end-user is asking to create the corresponding `Query`. See Section 6.15.2 and Figure 6.10 for an example of this presenter.

6.13.3 BUILDING YOUR OWN QUERIER

Writing Queriers follows similar principles to writing Presenters. To support UI code of any framework, the base `Querier` is a Java *interface*. The application developer needs to implement interaction code to determine what question the end-user is asking, and create the corresponding `Query`.

6.14 SELECTOR

The `Selector` component allows the Intelligibility Toolkit to formally support context-aware intelligibility, where question and explanation types are suppressed, provided, or promoted depending on contextual factors. These contextual factors are input as a `List` of `Parameters`. Each Selector encodes heuristics or design recommendations to appropriately provide explanations.

6.14.1 QUERY SELECTORS

We provide `QuerySelectors` to limit or promote questions for which to generate explanations given the input contexts. QuerySelectors operate before the `Querier` is rendered rather than after the explanation is generated, so as to restrict the end-users from asking question types that may not be suitable. After selection, a QuerySelector outputs a recommended `Query` or a list of Queries. We describe some QuerySelectors drawn from design recommendations of our user studies in other chapters. In this section, we refer to question types the way we refer to explanation types, emphasizing on the question asking capability rather than the explanation content.

SITUATIONQUERYSELECTOR

The `SituationQuerySelector` implements the design recommendations of Section 5.7 for which explanation types to show under various circumstances (*e.g.*, application behavior appropriateness, situation criticality, number of external dependencies). The selector starts with selecting no question types and adds corresponding question types that are necessitated by the contexts. Context values for these circumstances should be provided at design time or run time and supplied as inputs to the selector.

STREAMLINEQUERYSELECTOR

The `StreamlineQuerySelector` implements the streamline questioning recommendations of Section 7.10.3. It takes in the question type of the previously generated explanation at its input to determine what limited set of explanations may be shown, perhaps in a drop-down menu or button options.

CERTAINTYQUERYSELECTOR

The `CertaintyQuerySelector` implements the design recommendations in Section 8.12 for when to show intelligibility given the certainty of the application. It takes in the application's inference Certainty as input and gives a Boolean response of whether to show intelligibility. If the Certainty is below a threshold, say, 80%, then it will show a smaller subset of question types. On its own, this Selector does not specify any particular question type to suppress or provide.

6.14.2 REDUCER SELECTORS

Selectors may also be used to select which `Reducers` to apply to `Explanations` after they are generated by the `Explainer`. This allows for context-sensitive reduction or post-processing of explanations. ReducerSelectors output a recommended `Reducer` or list of `Reducers`. For example, explanations may be selectively reduced when shared with various contacts due to privacy concerns.

DETAILREDUCERSELECTOR

The `DetailReducerSelector` specifies the capability to choosing different `ConjunctionReducers` that reduce explanations to different lengths. It can be extended to follow different criteria, such as the contextual dependency of how much time a user may have to view the explanation. For example, in Section 9.7.2, we describe timing constraints for viewing intelligibility, and this directly impacts how much information end-users can view in the constrained times. Users are likely to have less time when they are unavailable than available. Hence, we can use inferred Availability as a context for whether to reduce more or less.

6.14.3 CHAINING OF SELECTORS

As can be seen, each Selector provides a different recommendation and each basis of selection are not mutually exclusive. Therefore, they can be chained together to apply multiple selection criteria. `QuerySelectors` chain their selections to the smallest subset of Queries they all select (*i.e.*, AND selection). `ReducerSelectors` chain their selections into a `CompoundReducer` that will apply multiple Reducers sequentially.

6.14.4 BUILDING YOUR OWN SELECTOR

We have presented a few Selectors that apply several criteria for selecting Queries or Reducers. All Selectors take in contextual inputs as a `List` of `Parameters`. `QuerySelectors` use criteria to select one or multiple Queries. QuerySelectors with new selection criteria need to satisfy the interface:

```
List<Query> select(List<Parameter> contexts);
```

Similarly, `ReducerSelectors` use criteria to select one or multiple Queries. ReducerSelectors with new selection criteria need to satisfy the interface:

```
List<Reducer> select(List<Parameter> contexts);
```

We can also have Selectors for other components in the Intelligibility Toolkit, namely, `Explainers` `Queriers`, and `Presenters`. These will implement the generic interface of `Selector`:

```
List<C> select(List<Parameter> contexts);
```

where `C` is the component type to be selected. As an example, we describe a `QuerierSelector`.

PROVISIONSELECTOR

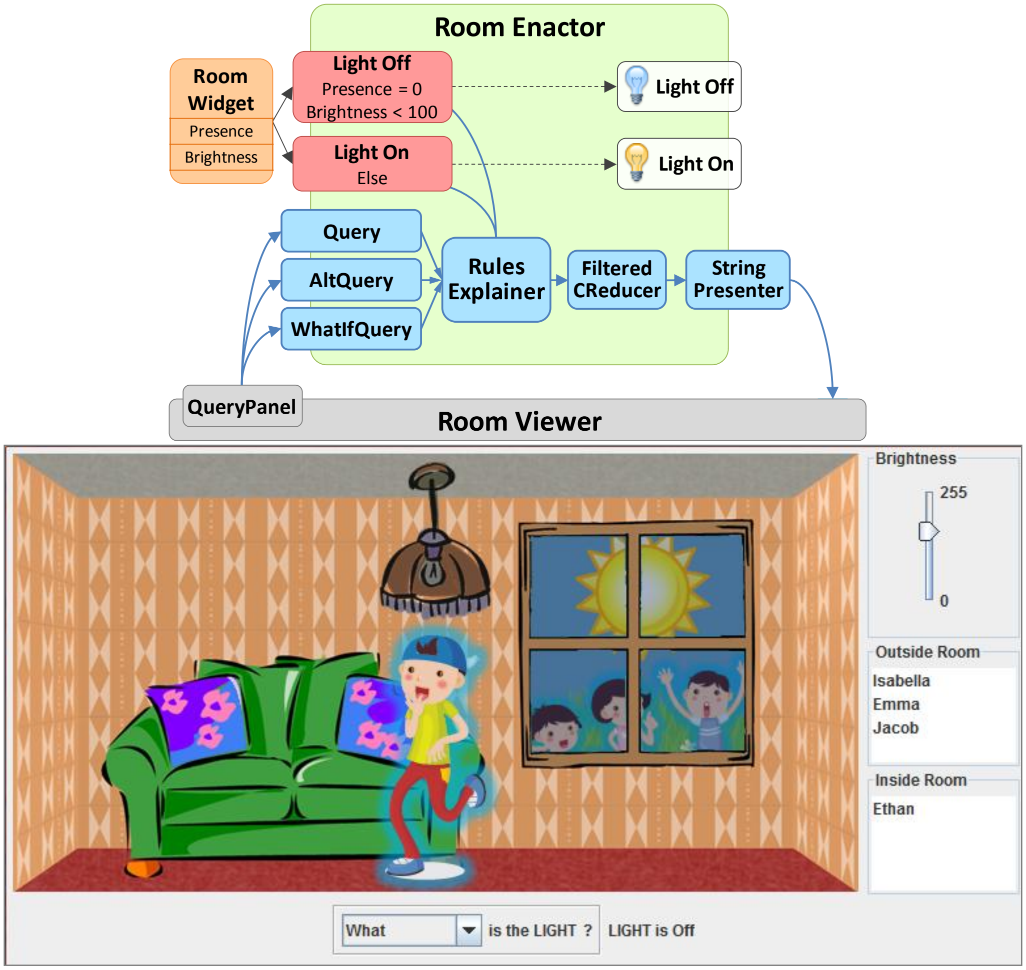
As a `QuerierSelector`, the `ProvisionSelector` selects the Querier interface component for presenting how a user may ask questions. Specifically, it specifies the capability to select whether to provide explanations On Demand (when the user explicitly asks), Automatic (context-dependent), or Always On (shown all the time). These explanation provision types are distinguished in [Gregor and Benbasat, 1999]. `ProvisionSelector` can be extended (or customized) to employ the appropriate provision type depending on the application needs. For example, the selector could switch from Always On to On Demand after the user has viewed the explanation more than a threshold number of times.

6.15 VALIDATION: DEMONSTRATION APPLICATIONS

To demonstrate that we can easily generate a range of explanations from context-aware applications using the Intelligibility Toolkit, we built three example intelligible applications. These examples demonstrate the use of the Intelligibility Toolkit for a span of *explanation types* and *model types*, and also cover a range of application domains for which the models are popular. While the toolkit significantly contributes to lowering the bar to providing explanations in context-aware applications, the explanations still need to be well designed (*e.g.*, for various UI, interaction, and device modality), and crafted specifically for each application problem domain. Therefore, rather than examining different explanation methods for the same model or application (*e.g.*, [Stumpf *et al.*, 2009]), we built different applications for each model type to *demonstrate the generality* of the Intelligibility Toolkit to provide explanations across application domains.

More examples and tutorials can be viewed and downloaded at the website we deployed for the toolkit at http://www.contexttoolkit.org.

6.15.1 Room Auto-Lighting – Rules



What ▾ is the LIGHT ? LIGHT is Off

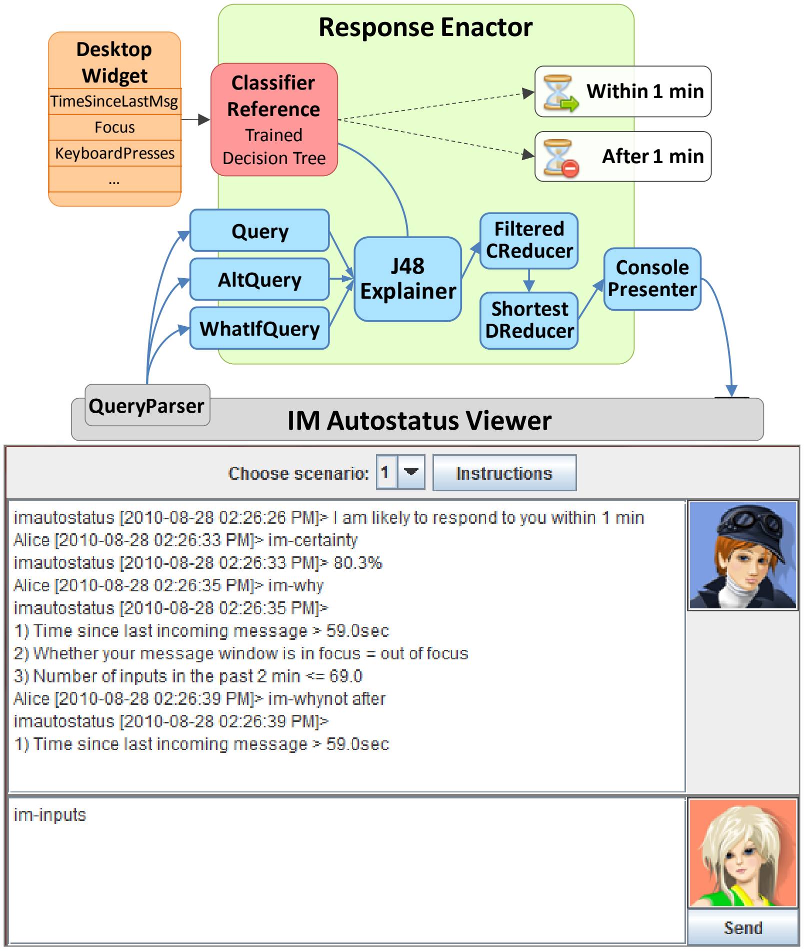
Figure 6.9: Automatic Room Lighting application demonstrating explanations of a rule-based application.

This demo application takes two factors (Presence and Brightness) to determine whether to turn the light on in a living room. The light would be off if Presence = 0 (i.e. no one in the room) or the detected brightness measure is less than 100 (out of 255). The application uses

- **QueryPanel** as a `Querier` to receive user interaction queries through a drop-down menu
- **RulesExplainer** to generate explanations from rules
- **FilteredCReducer** to filter out constant attribute terms in the rules
- **StringPresenter** to generate text output, which we simply display to a `JLabel`

We present a full tutorial on how to add intelligibility to a simplified version of this application in Appendix C.

6.15.2 IM AUTOSTATUS – DECISION TREE



```
imautostatus [2010-08-28 02:26:26 PM]> I am likely to respond to you within 1 min
Alice [2010-08-28 02:26:33 PM]> im-certainty
imautostatus [2010-08-28 02:26:33 PM]> 80.3%
Alice [2010-08-28 02:26:35 PM]> im-why
imautostatus [2010-08-28 02:26:35 PM]>
1) Time since last incoming message > 59.0sec
2) Whether your message window is in focus = out of focus
3) Number of inputs in the past 2 min <= 69.0
Alice [2010-08-28 02:26:39 PM]> im-whynot after
imautostatus [2010-08-28 02:26:39 PM]>
1) Time since last incoming message > 59.0sec
```

```
im-inputs
```

Figure 6.10: IM Autostatus demonstrating explanations from an instant messaging plug-in that uses a decision tree to predict when a buddy may respond.

We built an Instant Messaging plugin that predicts when a buddy will respond to a message (Figure 6.10). It is trained on an existing dataset from [Avrahami and Hudson, 2006] to build a decision tree. It takes desktop-based sensor inputs and makes response predictions (within, or after 1 min). The application uses

- **QueryParser** as a `Querier` to receive user interaction queries through text commands
- **J48RuleTraceExplainer** to generate rule trace explanations from the decision tree
- **FilteredCReducer** to retain only terms that are easily interpretably by end-users
- **ShortestDReducer** to select only the shortest reason if there are multiple reasons in the explanation
- **ConsolePresenter** which extends `StringPresenter` to generate text output and display it

We describe how we built this application in detail (the following two applications were built in a similar fashion) with the following procedure:

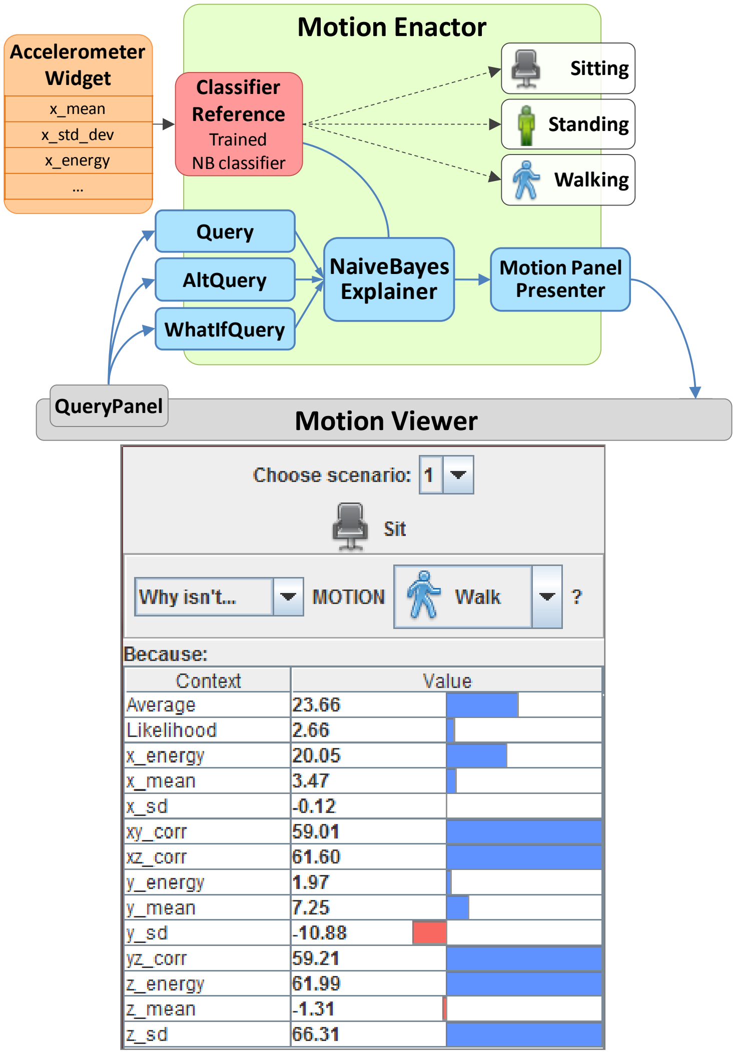
1. Create an `Enactor` for the overall application.
2. Create a `Widget` that tracks and updates all input features (extracted from the Subtle toolkit [Fogarty and Hudson, 2007]).
3. Set a pre-trained J48 decision tree classifier model to be the Enactor's classifier.
4. Set the Enactor's list of output values to two values: WITHIN_1_MIN and AFTER_1_MIN.
5. Create two `EnactorReferences`
    a. Associate them with the classifier and
    b. Associate one output value with one Reference.

   The developer implements what the application does when each Reference is triggered.
6. Create a `RulesExplainer` and associate with both References.
7. Set the `DisjunctionReducer` and `ConjunctionReducer` of the Explainer to `FirstDReducer` and `TruncationCReducer`, respectively.
8. Create an `IMAutostatusPresenter`, a custom extension of `RulesTextPresenter` that understands what each feature means to provide domain-specific textual explanations. It also handles printing an AIM message.
9. Code UI elements to invoke, on user prompt, various `getExplanation()` functions from the Explainer. The corresponding `Query` needs to be supplied when invoking each explanation type.

When the user asks for, say, a Why Not explanation about why not WITHIN_1_MIN:

1. The UI parses his request, populates an `AltQuerier` with WITHIN_1_MIN, and invokes `getExplanation(WhyNot, AltQuerier)`.
2. The Enactor takes the returned explanation `Expression` and passes it to `RulesTextPresenter` that renders it for the user as an IM response.

6.15.3 MOBILE MOTION RECOGNIZER – NAÏVE BAYES



Choose scenario: 1

Sit

Why isn't... MOTION Walk ?

Because:

Context	Value
Average	23.66
Likelihood	2.66
x_energy	20.05
x_mean	3.47
x_sd	-0.12
xy_corr	59.01
xz_corr	61.60
y_energy	1.97
y_mean	7.25
y_sd	-10.88
yz_corr	59.21
z_energy	61.99
z_mean	-1.31
z_sd	66.31

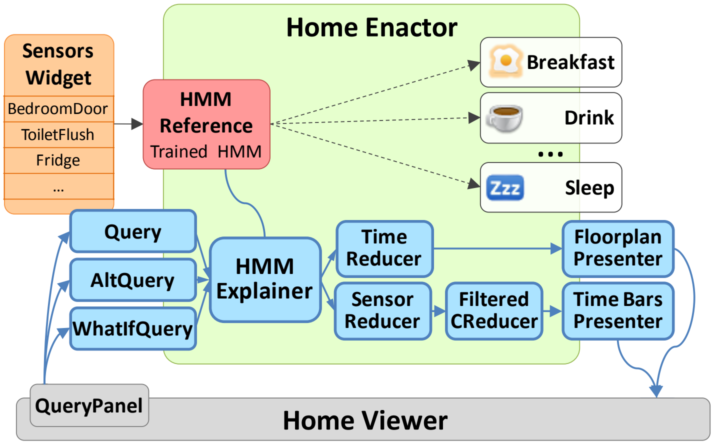
Figure 6.11: Motion Recognizer demonstrating a weights of evidence explanation for a mobile phone application inferring the physical activity of the user.

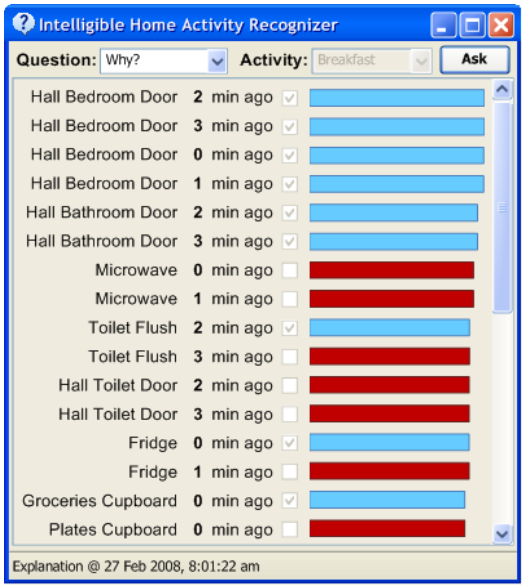
The naïve Bayes application we built is a physical activity recognizer that uses the accelerometer on a Google Android mobile phone to infer whether the user is sitting, standing, or walking (see Figure 6.11). It uses

- **QueryPanel** as a `Querier` to receive user interaction queries through a drop-down menu
- **NaiveBayesExplainer** to generate weights of evidence explanations from the naïve Bayes classifier

- No Reducer so that we can demonstrate how unreduced explanations may look overly technical to end-users. We recommend using **FilteredCReducer** to retain only terms that are easily interpretably by end-users
- **MotionPresenter** which extends `TypePanelPresenter` to render weights of evidence as a bar chart in a `JTable`

6.15.4 HOME ACTIVITY RECOGNIZER – HMM



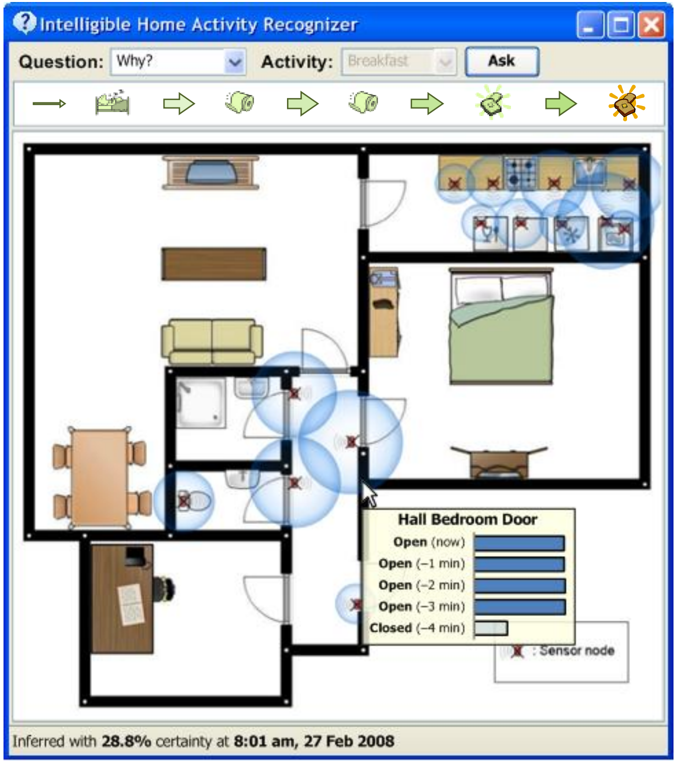


Intelligible Home Activity Recognizer

Question: Why? Activity: Breakfast Ask

Hall Bedroom Door	2 min ago	☑	🟦
Hall Bedroom Door	3 min ago	☑	🟦
Hall Bedroom Door	0 min ago	☑	🟦
Hall Bedroom Door	1 min ago	☑	🟦
Hall Bathroom Door	2 min ago	☑	🟦
Hall Bathroom Door	3 min ago	☑	🟦
Microwave	0 min ago	□	🟥
Microwave	1 min ago	□	🟥
Toilet Flush	2 min ago	☑	🟦
Toilet Flush	3 min ago	□	🟥
Hall Toilet Door	2 min ago	□	🟥
Hall Toilet Door	3 min ago	□	🟥
Fridge	0 min ago	☑	🟦
Fridge	1 min ago	□	🟥
Groceries Cupboard	0 min ago	☑	🟦
Plates Cupboard	0 min ago	□	🟥

Explanation @ 27 Feb 2008, 8:01:22 am



Intelligible Home Activity Recognizer

Question: Why? Activity: Breakfast Ask

Hall Bedroom Door

Open (now)	🟦
Open (–1 min)	🟦
Open (–2 min)	🟦
Open (–3 min)	🟦
Closed (–4 min)	□

Inferred with 28.8% certainty at 8:01 am, 27 Feb 2008

(Left) The bar chart visualization explains why the application inferred Breakfast in the last minute. Each row shows the weights of evidence for a sensor at a particular time step.

(Right) The floorplan visualization explains why the application inferred a sequence of

Sleeping → Toilet → Toilet → Breakfast → Breakfast, in the last 5 minutes.

Evidence due to features (summed across the last 5 min) are indicated by the area of bubbles around the corresponding sensors in the floorplan. Evidence for each sensor across time is revealed in a tooltip. We can see that the Hall Bedroom Door being open is a strong indicator of inferring the sequence. The door being open is a stronger indicator than it being closed 4 min ago. The microwave is another strong indicator (biggest bubble in top right corner).

Figure 6.12: Two Why visualizations for explaining a HMM to infer domestic activity.

We demonstrate explanations from an HMM model using the dataset from [van Kasteren *et al.*, 2007] about domestic activity, and train a HMM with a sequence length of 5 min, and 1 min per sequence step. The application takes 14 binary input sensors and infers which activity (out of seven) the user is performing. It uses

- **QueryPanel** as a `Querier` to receive user interaction queries through text commands
- **HmmExplainer** to generate two-dimensional (inputs and time) weights of evidence explanations from the HMM
- **TimeReducer** which extends `DimensionReducer` to aggregate the weights of evidence along the dimension of time, showing weights of evidence due to input features
- **SensorReducer** which extends `DimensionReducer` to aggregate the weights of evidence along the dimension of input features, showing weights of evidence due to time
- **FilteredCReducer** to select weights of evidence relevant to the input feature over time (for a tooltip display)
- **FloorplanPresenter** to display the floorplan overlaid with bubbles, where their size represents the weight of evidence due to the input feature over time
- **TimeBarsPresenter** to display a bar chart visualization, where each bar represents the weight of evidence due at a specific time

6.15.5 LAKSA — AVAILABILITY INFERENCE MOBILE APPLICATION

We have developed, Laκsa, a high-fidelity prototype of an intelligible context-aware mobile application that further validates the Intelligibility Toolkit. It uses a version of the toolkit ported to Android to generate explanations from rules, a decision tree, and naïve Bayes to explain inferences of availability, motion activity, and sound activity, respectively. Section 7.2 describes the first version of Laκsa with its intelligibility UI implemented in Java Swing for a usability study prototype, and Section 9.3 describes the second version of Laκsa with its interface fully developed in Android 2.2.

6.16 LIMITATIONS AND DISCUSSIONS

While the Intelligibility Toolkit is extensible, the current implementation does not cover some outstanding aspects.

i. There remain some types of explanations that users ask for that are not yet supported: *e.g.*, how to Control an application to change its behavior (though this is supported with Parameters in the Enactor framework [Dey and Newberger, 2009]), and Provenance (source, credibility, and accuracy of the application inputs).

ii. Often, sensed raw inputs of context-aware applications are pre-processed, *e.g.*, using signal processing or computer vision techniques. Although important, the toolkit does not currently capture and explain these pre-processing mechanisms [Patel *et al*., 2008].

iii. Similarly, the Intelligibility Toolkit focuses on explaining the inference of the application model when it encounters situations, but not how the model was originally trained, or how it continues to adapt itself (if using active learning). This may help end-users understand how the training data may influence the application's behavior, but may also pose a cognitive burden on end-users to understand data-driven models.

iv. Applications may encounter behaviors due to the infrastructure rather than the decision model or inputs. For example, unexpected behavior may result from resources suddenly being unavailable due to connectivity issues. The toolkit does not currently support explanations about the infrastructure.

v. The Intelligibility Toolkit provides components to facilitate the development of intelligible applications, the automatic generation of explanations, and mechanisms for presenting the explanations. However, it does not provide guidelines on how to present the explanations.

vi. On a related note, this work did not seek to evaluation of the impact or effectiveness of explanations that may be generated from the Intelligibility Toolkit, but provides tools for such investigations in the future.

6.17 RELATED WORK IN EXPLAINING CONTEXT

The Intelligibility Toolkit supports a wide range of explanations for multiple decision models. Previous systems only covered a subset of the explanation types, and only for one or one type of decision model.

The most similar framework to our toolkit is the Enactor framework [Dey and Newberger, 2009], on which we base our Intelligibility Toolkit. It can provide What explanations by exposing the state of input Widgets, and Why explanations by reporting a relevant rule. However, it does not support the other explanation types or any models beyond rules. The Crystal framework [Myers *et al*., 2006] supports only Why / Why Not explanations for desktop-based applications to explain themselves through Command Objects. The Whyline [Ko and Myers, 2009] similarly explains Why / Why Not to end-user programmers, by examining the program execution tree. PersonisAD [Assad *et al*., 2007] defines a distributed framework to support What explanations by resolving identities and associations of devices, locations, people, *etc*. The Intelligent Office System [Cheverst *et al*., 2005] provides What explanations by showing the system state, History explanations by listing the states across time, and Why explanations about the learned cut-points for its rules. Panoramic [Welbourne *et al*., 2009] provides Why, What, and History explanations to explain location events through a visualization of parallel timelines of sensed and rule-determined events.

While the aforementioned systems provide explanations for rules, Tullio *et al*. [2007] explained interruptibility inferred from decision trees and naïve Bayes with What explanations. The Intelligibility Toolkit can provide deeper (*e.g.*, Why, Why Not, How To) explanations from these models. Kulesza *et al*. [2009] built an intelligible email sorter that uses naïve Bayes for classification. It provides Why, Why Not, and What If explanations based on the weights of evidence approach [Poulin *et al*., 2006]. The Intelligibility Toolkit also uses this approach, and adds more explanation types, supports numeric input features, extends it for HMMs, and has been developed to be extensible.

6.18 CONCLUSION AND FUTURE WORK

We have presented the Intelligibility Toolkit that currently provides automatic generation of 12 explanation types for at least 10 popular inference models in context-aware applications. It supports

i. Generating explanation structures with **Explainers**,
ii. Querying mechanisms to specify questions and constrain explanations with **Queries**,
iii. Representing explanations with **Explanation Expressions**,
iv. Simplifying complex explanations with **Reducers**,
v. Presenting the explanations to end-users and other subsystems with **Presenters**,
vi. Presenting interfaces to ask questions with **Queriers**, and
vii. Providing context-sensitive intelligibility with **Selectors**.

Explanations can be provided as either *rule traces* or *weights of evidence*. The toolkit is also extensible to support new explanation types, model types, reduction heuristics, and presentation formats.

The Intelligibility Toolkit aims to make it easier for developers to provide many explanation types in their context-aware applications. This ease also allows developers to perform rapid prototyping of different explanation types to discern the best explanations to use and the best ways to use them. By standardizing the styles of explanations, developers have many more choices when choosing classifiers to increase application accuracy and performance, while retaining the intelligibility features of their application, and not even needing to change explanation interfaces (especially so for weights of evidence explanations).

In addition to addressing the limitations outlined earlier, we can use the Intelligibility Toolkit, to pursue further research questions regarding the intelligibility of context-aware applications. In particular, we can investigate and compare the efficacy of various explanation types, by measuring how well each type helps users to understand the application, and improve their trust in the application. Using the Intelligibility Toolkit, we developed an intelligible context-aware mobile application (described in Chapters 7 and 9) with multiple explanation types, and multiple context types (*e.g.*, location, physical activity, sound activity), and conduct a usability and usage evaluations of the impact of intelligibility and how well intelligibility improves understanding or corrects misunderstanding.

7 DESIGNING FOR INTELLIGIBILITY

This chapter is an extension of the work presented in:

> Lim, B. Y. and Dey, A. K. (2011). Design of an Intelligible Mobile Context-Aware Application. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI '11)*. ACM, New York, NY, USA, 157-166.

ABSTRACT. Context-aware applications are increasingly complex and autonomous, and research has indicated that explanations can help users better understand and ultimately trust their autonomous behavior. However, it is still unclear how to effectively *present* and *provide* these explanations. This work builds on previous work to make context-aware applications intelligible by supporting a suite of explanations using eight question types (*e.g.*, Why, Why Not, What If). We present a formative study on design and usability issues for making an *intelligible* real-world, mobile context-aware application, focusing on the use of intelligibility for the mobile contexts of availability, place, motion, and sound activity. We discuss design strategies that we considered, findings of explanation use, and design recommendations to make intelligibility more usable.

7.1 INTRODUCTION

In Chapter 6, we had addressed the support of intelligibility at the underlying level by automatically generating explanation types. In this chapter, we shift our focus to the user interface level of providing intelligibility in context-aware applications. Indeed, there have already been several context-aware applications that support some level of intelligibility. Cheverst *et al.*'s Intelligent Office System [2005] exposes sensor values, and explains its fuzzy decision tree model that controls office appliances. Tullio *et al.*'s interruptibility displays [2007] explain how they determine a manager's interruptibility by exposing the values of sensors in the manager's room. Kulesza *et al.* built an email sorting application [2009] that supports Why (*i.e.*, why the application took a particular action) and Why Not (*i.e.*, why the application did not take a different action) explanations for its naïve Bayes classifier. Vermeulen *et al*.'s PervasiveCrystal [2010] also supports Why and Why Not explanations but for ambient environments. These systems support a limited set of explanations users can ask for: What, Input values, Why, and Why Not. However, we found that users ask a wider range of questions of context-aware applications (Chapter 5), and that different explanations have different impacts on user understanding (Chapter 4).

We had developed the Intelligibility Toolkit to automatically support this wider range of explanations. While this significantly helps facilitate the development of intelligible context-aware applications, it remained unclear how to effectively *present* these explanations. In this chapter, we advance the knowledge of how to design for intelligibility by investigating explanation design for a real-world, mobile context-aware prototype. We focused on intelligibility for the mobile contexts of availability, place, motion, and sound activity.

Our contributions in this chapter are the:

1. Exploration of *design* and *usability* issues in making a context-aware application intelligible, and
2. Provision of design recommendations to address them.

The rest of the chapter is organized as follows: we describe the prototype that we developed to be intelligible. We then describe our rationale for designing the explanations to make them more interpretable. To discover how users use intelligibility and the usability issues that they face, we ran an *in-situ*, scenario-driven, think-aloud study. We describe our experimental set up, method and data analysis. We then describe how our participants used explanations, and discuss our interpretations of some factors influencing their behavior. We observed how participants used explanations differently depending on their goals, and how some participants encountered difficulties due to their lack of prior knowledge or their chosen problem solving strategies. Finally, we provide design recommendations arising from these observations, and describe our plans for future work.

7.2 LAKSA — SOCIAL AWARENESS APPLICATION

Staying aware of others is an established need that people have [Oulasvirta, 2005]. Between close friends and family members, this can help people feel a greater sense of connectedness as each person goes about their daily activities. Between coworkers, this can inform people of the most suitable times to contact them. We have developed Laκsa, a *mobile* application that shares people's availability status. Laκsa is an acronym for Location, Activity, Connectivity (κ), and Social Awareness that describe its function. Availability is determined from six lower-level contexts: Place, Motion, Sound activity, phone Ringer, Schedule, and who is enquiring (Contactor). While similar to CenseMe [Miluzzo *et al.*, 2008], it uses a slightly different set of contexts, aggregates them into an availability context through rules (rather than just showing all contexts as *presence* information), and is intelligible, such that it can explain its complex behavior.

Our focus in this chapter is the use of Laκsa as a vehicle to explore the design, implementation, and use of intelligibility in a sophisticated, multi-factor context-aware application. In the rest of this section, we shall describe the various contexts that Laκsa employs, briefly describing their implementation. Laκsa is designed to support the complexities of availability in social relations by considering availability as *multi-faceted*, and *multi-factored*. One's availability depends on who is asking (different *facet* for different viewer), and on contextual *factors* (*e.g.*, Place, Motion, Sound). Figure 7.1 describes Laκsa's context hierarchy. We designed Laκsa to be sophisticated in using many contexts, and complex sensing and inference mechanisms, to exemplify how context-aware applications can manage many factors that users would find cognitively difficult. We anticipate users will ask for explanations to determine or remember Laκsa's complex mechanisms. Next we describe the contexts Laκsa models.

Top-tier Context	Availability	Intelligibility Explanations
Lower-tier Contexts	Place, Motion, Sound, Ringer, Schedule, Contacter	
Lower-tier Input Features	Latitude, Longitude; Energy, Mean, Standard Deviation; MFCCs, *etc*.	
Sensors and Sources	GPS, Wi-Fi; Accelerometer; Microphone; Phone State, Calendar, *etc*.	

Figure 7.1. Laκsa context architecture with different tiers of context used to infer higher-level tiers. The user sees the Availability status, and the intelligibility explanations.

Availability: *Available, Semi-Available, Unavailable* — is determined based on rules regarding the following six factors. The contactor interprets the availability and decides whether to contact and how (call, text, email, *etc*.).

Contactor (Who is Enquiring): *Family, Friend, Coworker,* and *Default* (to check user's default status) — categorizes person contacting or enquiring about the user.

Place: *Home, Office, Café, Library, etc.* — represents the semantic location of the user. It is computed by sensing latitude and longitude from the Skyhook Wi-Fi API (uses a hybrid GPS, Wi-Fi, and cell tower positioning algorithm), and matching to a pre-determined named location that the user specifies. To convey accuracy, it also reports the sensed distance error and detected number of access points.

Motion: *Sitting, Walking, Cycling, Placing the phone Flat*, *etc.* — represents the user's physical activity inferred with the phone placed in a front pants pocket. Inferences are made with a decision tree trained using activities from several users. Features extracted from the accelerometer are similar to [Bao and Intille, 2004; Lester, Choudhury, and Borriello, 2006]: *e.g.*, mean and standard deviation for three axes, phone orientation angles, and signal powers.

Sound: *Talking, Music, and Ambient Noise* — represents the sound activity that Laκsa recognizes from what it can hear from the phone's microphone. Inferences come from a naïve Bayes classifier trained on sound samples. Features extracted are similar to [Lu *et al.*, 2009]: *e.g.*, mean and standard deviation of power, low-energy frame rate, spectral flux, spectral entropy, spectral centroid, bandwidth, Mel-Frequency Cepstral Coefficients (MFCCs).

Phone Ringer: *Silent, Vibration, or Normal* — represents the volume state to which the phone ringer has been set.

Calendar Schedule: *Personal, Work, or Unscheduled* — represents the user's Google Calendar, and, in particular, which calendar the current event is in.

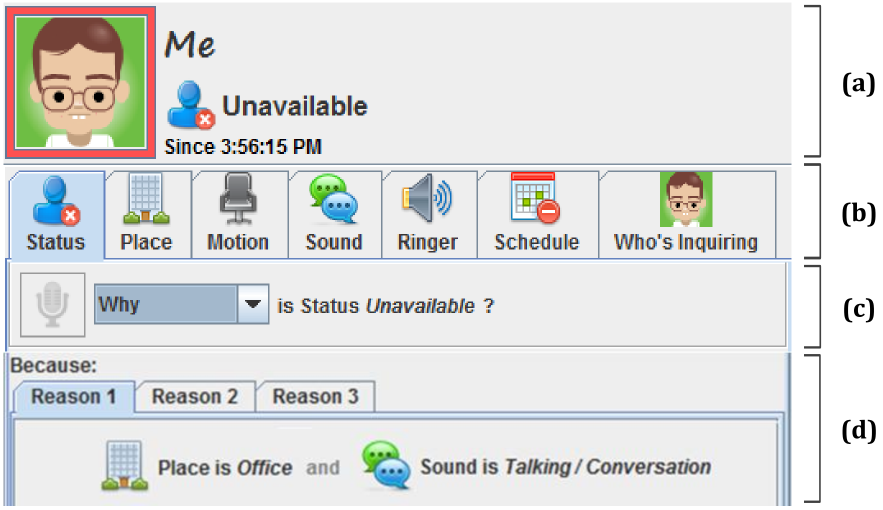
While one could compare the use of explanations for different contexts (*e.g.*, Place, Motion, Sound) and different decision models (rules, decision trees, naïve Bayes), for this formative work, we focus on exploring the design and use of explanations across this breadth of factors.

7.3 DESIGN OF INTELLIGIBILITY

Having defined the context types, we seek to make Laκsa intelligible so that users can understand what it knows and how it makes decisions about user availability. We employ explanations supported by the Intelligibility Toolkit (Chapter 6):

1. **What** is the value of the context?
2. **Why** is this context inferred as the current value?
3. **Why Not**: why isn't this context inferred as Y, instead?
4. **How To**: when would this context take value Y?
5. **Inputs**: what details affect this context? (Factors, input features, related details, *etc.*)
6. **Outputs**: what other values can this context take?
7. **What if** the conditions are different, what would this context be? (Requires user manipulation)
8. **Certainty**: how confident is Laκsa of this value?
9. **Description**: meaning of the context terms and values.

Explanations generated from the Intelligibility Toolkit contain the information content to answer these nine questions and can be rendered using simple text templates. However, these may not be easy for lay users to interpret or quickly assimilate. Therefore, we employed several design strategies to help make the explanations more usable. Figure 7.2 and Figure 7.3 show some explanation user interfaces resulting from the strategies described next.



Me

Unavailable

Since 3:56:15 PM

Why is Status *Unavailable* ?

Because:

Place is *Office* and Sound is *Talking / Conversation*

a) This is the core information to show the Availability status that Laκsa has inferred. It shows the user or buddy's picture, name, and Availability status. It also shows the time since the last change.

b) To ask for an explanation, first select which context you would like to ask about by selecting one of these seven Context Tabs. Each tab shows the current value of the context (What explanation), *e.g.*, Sound = Talking.

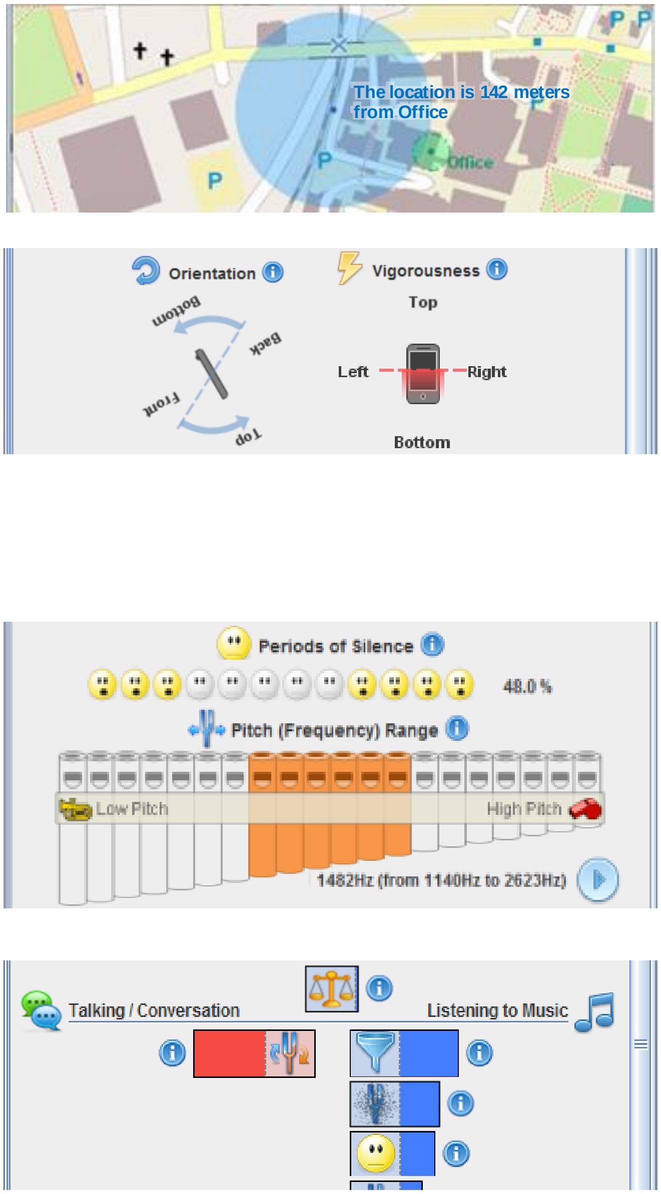
c) Next, you ask a specific question from the drop-down menu. The Question Panel adapts to the question selected (*e.g.*, Why, Why Not, What If).

d) The resulting explanation is rendered in the Explanation Panel. **List** visualization: the left screenshot shows a reason with a list of multiple factors. Some explanations have multiple alternative reasons, shown in different sub-tabs.

Figure 7.2. Screenshot of the Laκsa showing how to use the components in the core and intelligibility user interface.

7.3.1 REDUCING AND AGGREGATING EXPLANATIONS

We expect that users of context-aware applications would rather be focused on their day-to-day tasks than dedicate too much attention to technical details. Hence, it is important to simplify and *reduce* the provided explanations to be concise and salient. We have done this for Laκsa by aggregating explanation types (*e.g.*, What value and Certainty rating shown together), reducing the number of reasons, length of reasons, and number of input features (*e.g.*, omitting MFCCs for sound); and combining explanations for simultaneous consumption (*e.g.*, presenting x-y-z accelerometer values in 2D diagrams). While this may compromise comprehensiveness, it is intended to make the explanations more interpretable.



(a) Map visualization explaining with 'bubble' components. Blue bubble indicates estimate of current location; green dotted bubble indicates specified region of named location.

Why is my Place inferred as "Office"? Because your estimated location bubble overlaps with the bubble specifying where Office is.

(b) Physical Diagram visualization illustrating the interpreted mechanical motion of the phone leading to its inference of cycling (two shown here). Dotted lines and arrows or shading show boundary conditions of rules of the decision tree model. Users can mouse over the diagram to see textual explanations with numerical values.

Why isn't Motion inferred as "Cycling"? Because the phone is oriented more than the drawn angle, the vigorousness detected is below the illustrated value, *etc*.

(c) Metaphorical visualization of sound feature values that was sensed and computed at a specified moment (two shown here). This can show *current* values sensed, or *average* values for each possible outcome.

What details (Inputs) affected the Sound inference? 48% of the sound heard was relatively silent (Periods of Silence); the range of pitches heard was from 1140 to 2623Hz (this can be played aurally for the user to listen to), *etc*.

(d) Weights of Evidence visualization. Bar chart visualization showing weights of evidence voting for or against the inference of Talking. Weights are represented by the length of each bar; color and positioning indicate evidence for or against. Not all factors shown.

Why isn't Sound inferred as "Talking"? Because most of the factors vote for the inference of Music (especially hidden features, the sound Volume, Periods of Silence, *etc*); only Pitch Fluctuation votes for Talking.

Figure 7.3. Explanation Visualizations rendered in the explanation panel. Some examples and their interpretations. The UI uses icons derived from [Fatcow].

7.3.2 VISUALIZING EXPLANATIONS

Users should more quickly assimilate visual explanations because of the higher bandwidth of diagrams [Ware, 2000]. Hence, we provide several visual representations: icons for context and feature values, dynamic diagrams that change when values change, and even animation and sound (to hear pitches and pitch ranges). Visualizations are customized for the context domains (*e.g.*, map for Place, physical 2D phone diagram for Motion). Since Sound is not visual, we chose to explain sound by metaphor (*e.g.*, showing a pan flute to represent pitches and ranges; see Figure 7.3c).

7.3.3 EXPLAINING IN SIMPLE AND RELATABLE TERMS

As we deploy intelligibility in real-world prototypes, users need to understand how the contexts and recognition features relate to the real world, using lay concepts. Therefore, we simplify names (*e.g.*, "spectral entropy" renamed to "pitch pureness", "low-energy frame rate" renamed to "periods of silence"), normalize numerical values to lay scales (*e.g.*, 0 to 100), and include Description explanations that describe what each context factor and feature mean. Descriptions are presented in physical rather than system terms (*e.g.*, there are more *periods of silence* for talking than ambient noise, because there are significant pauses in speech that are relatively quiet). Furthermore, features for Motion and Sound were scaled to physically meaningful names and values (*e.g.*, *vigorousness* to represent accelerometer signal power in *Watts*). We also chose to visualize explanations for motion using *first principles*. For example, orientation information is shown as the orientation of the phone relative to the ground (see Figure 7.3b). Note that ensuring that terms are domain relatable may require significant domain knowledge, rather than just naïvely applying effective features identified in the literature about activity recognition (*e.g.*, [Lester, Choudhury, and Borriello, 2009; Lu *et al.*, 2009]).

7.3.4 PROVIDING EXPLANATIONS FOR CONTROL

We chose to provide explanation types for each context only if users could leverage the information to improve Laκsa, or change their behavior. What If explanations are only provided for the top-tier Availability context. For other contexts, users would not be able to meaningfully change the input features (*e.g.*, changing the entropy or frequency to influence sound). Furthermore, we omit trivial explanations, *e.g.*, asking What If one is at a specific coordinate location to learn which semantic place Laκsa would infer the user being at; asking Why Not questions about manually set contexts (*e.g.*, schedule or ringer mode).

We iterated on the design of Laκsa with these strategies and feedback from colleagues who are active HCI researchers.

7.4 LAKSA PROTOTYPE IMPLEMENTATION

We built Laκsa using a client-server architecture. Sensing of low-level contexts (*e.g.*, latitude, longitude coordinates, accelerometer, microphone) was performed on an Android mobile phone, and some intermediate features (*e.g.*, discrete Fourier transform, entropy, energy) were computed on the phone. The extracted features are then sent via XMPP to a server for further processing. On the server, we modeled the contexts for users with the Enactor framework [Dey and Newberger], and used the Intelligibility Toolkit [Lim and Dey, 2010] to support the *querying* for various questions, *generation* and *reduction* of explanations about the contexts, and *presentation* of the explanations in various graphical and textual formats.

To measure how participants use the Laκsa interface, and to support rapid prototyping, we developed the interface on a touch screen tablet, rather than on the mobile phone. Users interact with the UI with a mouse, pen stylus, or finger. The latter interaction closely resembles cell phone interaction.

7.5 LAKSA PROTOTYPE USAGE

Users ask for explanations by selecting a question from the drop down menu (see Figure 7.2a). Each question only pertains to the particular context of focus (*e.g.*, Availability). To ask about another context, the user selects the context by its tab (Figure 7.2b), and asks the questions in the panel (Figure 7.2c). The resulting explanation appears in the Explanation Panel (Figure 7.2d; Figure 7.3; Table 7.1).

	Description / Function	Availability	Place	Motion	Sound
What	Shows current value / state of context.	Value	Map (location bubble)	Value	Value
Why	Shows a model-based explanation of how Laκsa inferred the current output value (outcome).	List (multiple conditions)	Map (user & actual place bubbles *overlapping*)	Physical Diagram (with boundary conditions)	Weights of Evidence
Why Not	Shows a model-based explanation distinguishing how Laκsa did not infer the alternative output value.	Multiple Lists (multiple conditions)	Map (user & desired place bubbles *separated*)	Physical Diagram (with boundary conditions)	Weights of Evidence
How To	Shows a model-based explanation of how Laκsa typically or generally infers the target output value.	List (required conditions)	Map (actual place bubble)	Physical Diagram (of *average* input values for desired outcome)	Metaphorical Viz. (of *average* input values for desired outcome)
Inputs	Shows the current values of input context / features.	List (current input values)	List (latitude, longitude)	Physical Diagram (of *current* input values)	Metaphorical Viz. (of *current* input values)
Outputs	Shows possible output values the context may take.	List (*possible* output values)			
What If	Shows a UI to allow the user to specify different input values and see what the output value would be.	Editable List (of current input values)			
Certainty	Shows the certainty or confidence of Laκsa ‘s inference of the current context value.		List (distance error, # access points)	% Certainty (of inference)	% Certainty (of inference)
Description	Shows terminology or description of context / feature.	Text (sentences)			

Table 7.1. Explanation Visualization Types for each context, and question type. Only the What value is shown for Ringer, Schedule, and Contactor. Note that Certainty is shown together with both What

7.6 Scenario-Driven Think Aloud User Study

To explore the use of the intelligibility features in Laκsa, we conducted a *scenario*-driven user study where participants *think aloud* as they used it. This was an exploratory study where we investigated how and why participants used intelligibility, and how this use impacts their understanding of Laκsa. We conducted an *in-situ* controlled study rather than a field deployment for two main reasons: (i) to present participants with a lower-fidelity interface to elicit more feedback from the think aloud study, and (ii) to avoid having serious usability issues that could confound results in a field study, where it would also be harder to monitor participants' usage and rationale.

7.6.1 Procedure

The experiment began with the first scenario (see later) as a training session, where the experimenter explained the function of the Laκsa prototype as an availability awareness application, its sensing capabilities, and its intelligibility features. Participants verified that they understood the interface and explanations by stepping through the interface themselves and thinking aloud what they understood Each subsequent scenario involved participants moving around the university campus to a respective location (*e.g.*, library, café, office) and engaging in a specific activity (*e.g.*, walking, cycling). An experimenter shadowed the participants all the while. The participants were then told of an *incident* (*e.g.*, having one's phone ring loudly while searching for a book in the library) and the phone's subsequent behavior. They were asked for their opinion of the situation, and of the behavior of the application. They were then asked to explore Laκsa to find out what is really happening or to clarify the situation. The participants were prompted to think aloud as they did this, and the experimenter probed for clarification. For each scenario, we recorded what participants did and said during the think aloud. Finally, we interviewed them about their opinion and understanding of how Laκsa sensed and inferred contexts.

7.6.2 Controlled In-Situ Scenarios

The user study was scenario-driven to expose participants to a wide range of situations they may encounter with Laκsa. To increase the visceral quality of the scenarios, we engaged the participants in actually physically performing the tasks *in-situ*, rather than just imagining themselves in the respective environments and situations. For example, we had participants go into a library to look for a book (S4 below), and ride a stationary bicycle (S5). To better control conditions, we *simulated* the sensor data that Laksa used for each scenario. The data is based on previously recorded real data of several users performing the respective scenarios. We asked participants to wear pants with front pockets to facilitate placing the mobile there and to allow comfortable cycling. To further control for experience, we provided participants with a fixed set of personal availability rules (see Table 7.2).

Availability	Location	Motion	Sound	Ringer	Schedule	Who's Enquiring
Unavailable	Office			Silent		*Anyone*
Unavailable	Office		Talking			*Anyone*
Unavailable					Work	Friends
Semi-Available	Library					*Anyone*
Semi-Available	Home					Coworkers
Semi-Available		Cycling				*Anyone*
Available	*All other cases*					

Table 7.2. Personal availability rules participants were told were pre-programmed into Laksa for the user study scenarios.

We employed seven scenarios to span three situational dimensions: (i) Exploration / Verification (S1, S2, S5) where Laksa behaved appropriately and participants are asked to explore and verify the interface; (ii) Fault Finding (S3, S4, S7) where Laksa apparently or actually behaved inappropriately, and participants had to debug what really happened; (iii) Social Awareness (S6, S7) where participants investigated information and explanations about hypothetical buddies that they naturally had less awareness of. We measured their desire to contact their buddy in the latter scenarios to gauge their trust of Laksa.

S1: Sitting in office talking. Training session where the participant learned Laksa's core features and explanations.

S2: Walking outdoors. Verification/exploration task where participants investigated explanations of Place and Motion.

S3: Missed contact. The participant was asked to find out why she was considered not available to a friend, even though she was. This is a false-positive error condition where Laĸsa actually behaved correctly, but given that availability is multi-faceted *(i.e*. different statuses to different viewers), the participant may not realize this.

S4: Library interruption. The participant walked to a nearby library to search for a specific book. She needed to squat to retrieve the book on the lowest shelf. As she was looking for the book, the phone rang (the experimenter invoked a ringtone in the library), indicating a call from a coworker. The participant was asked to determine why she was not seen as unavailable. This is a true-positive error condition where we simulated Laĸsa making a mistake.

S5: Cycling in gym. Exploration task for participants to explore how Laĸsa tracks their cycling in a gym location.

S6: Friend available. The participant was asked to check Laĸsa to help decide whether to contact a friend.

S7: Friend talking inferred as music. The participant was asked to find out why, when the friend was actually in a meeting, Laĸsa made the error of telling her that the friend was Available, at the office listening to music, and had nothing scheduled.

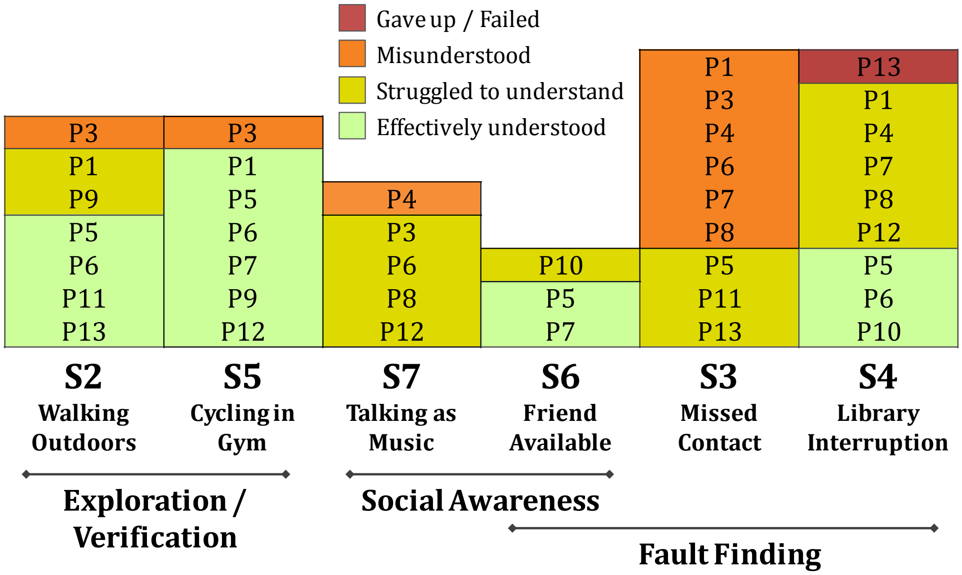
7.6.3 PARTICIPANTS

Using a local recruiting website, we recruited 13 participants (8 females) with a mean age of 26.4 years (range: 18 to 37). P2 was dropped because he did not continue beyond the training scenario. Four participants were students (one undergraduate). Only P1 was trained in a computer-related field (information systems), while the others spanned a wide range of areas (*e.g.*, rehabilitation, mathematics, materials science and engineering, human resources). We engaged each participant for 2.5 hours on average (range: 2 to 3.5). Due to the length of each scenario, each participant experienced between 3 and 6 scenarios (median=4), selected to try to balance coverage. Participants were compensated $10/hr.

7.7 DATA ANALYSIS

We transcribed the think aloud and interview data, segmented by speaker (interviewer / interviewee). The transcript was coded by question type (*e.g.*, Why, Why Not, What If), goal / intention / rationale (verified during interviews), feature requests, breakdowns / struggles (*e.g.*, too many questions to choose from), and extent of (mis)understanding. We formed sequence models of the usage of question types, and consolidated them (*e.g.*, see Figure 7.4). We interpreted the findings and models to identified causal factors. This was assembled into higher-level themes using an affinity diagram (selection based on theme convergence, importance, and novelty).

Next, we describe our observations of how participants used Laĸsa, focusing on how they asked for various explanation types. Table 7.3 summarizes how much participants understood Laĸsa's inference in each scenario.



	S2 Walking Outdoors	S5 Cycling in Gym	S7 Talking as Music	S6 Friend Available	S3 Missed Contact	S4 Library Interruption
🟥 Gave up / Failed						P13
🟧 Misunderstood	P3	P3	P4		P1 P3 P4 P6 P7 P8	
🟨 Struggled to understand	P1 P9		P3 P6 P8 P12	P10	P5 P11 P13	P1 P4 P7 P8 P12
🟩 Effectively understood	P5 P6 P11 P13	P1 P5 P6 P7 P9 P12		P5 P7		P5 P6 P10

Exploration / Verification
Social Awareness
Fault Finding

Table 7.3. Participant results in scenarios showing how each participant performed for each scenario as he or she used explanations provided in Laĸsa. Note that columns are arranged in categories, not by sequence of presentation.

7.8 FINDINGS — PATTERNS OF INTELLIGIBILITY USE

We found different usage of explanations for the three different situational dimensions presented in the scenarios: exploration / verification, social awareness, and fault finding.

7.8.1 EXPLORATION / VERIFICATION

We observed that participants explored all explanation types as they tried to *learn* how Laĸsa functioned, and more about the scenarios. Naturally, participants used the Description explanations to *remind* them what the terms and concepts meant, and used the Inputs explanation to *examine* deeper states that affect inferences (*e.g.*, for S3, P11 used the Inputs explanation of Availability to see a “summary of her statuses”). How To explanations were used in two ways: P11 appreciated learning new concepts about inferring Sound (through periods of silence, pitch ranges, *etc.*); for S3, P6 checked to see when she would be Unavailable. Finally, some participants asked What If to *preemptively* test troublesome or critical situations to see how Laκsa would respond under those circumstances; *e.g.*, for S3 during a lull period, P3 and P5 confirmed that they remained available to family members in an emergency.

7.8.2 SOCIAL AWARENESS

For S6 and S7, participants had less *first-hand* knowledge of the actual situation about their buddy than about themselves. We observed that they mainly focused on the Input explanation of Availability (or equivalently, What explanations of the lower-level contexts). For example, P6 first checked the Input values of Availability to determine the state of her friend (in S7). Participants would then form *stories* about what they believed their buddy could be doing at the time; *e.g.*, having seen the Motion inferred as “Other” (placed flat) for S6, P9 presumed his friend was “probably sleeping or taking a nap or eating or something that would probably involve not wanting a phone call.” When he subsequently looked at the Sound inference, he said “sound is 78%, oh he is listening to music, so I guess I could intrude if I want to, but then I get he might be with someone else too.” At this point, most participants did not continue by asking other questions. However, for S6, P7 asked When would her friend be Semi-Available (How To) to learn the rules her friend had set.

We found that once participants perceived that the availability status was inferred inaccurately or erroneously, they explored other questions and investigated more deeply. This is similar to how participants investigated anomalies with explanations about themselves (discussed next).

7.8.3 FAULT FINDING

Participants used explanations most when perceiving that Laκsa behaved *unexpectedly*. Figure 7.4 summarizes the sequences of how participants asked for various explanations (labels refer to observations described in the following text). The choices of questions were slightly different when investigating the top-tier Availability than the lower-tier contexts (Place, Motion, and Sound).

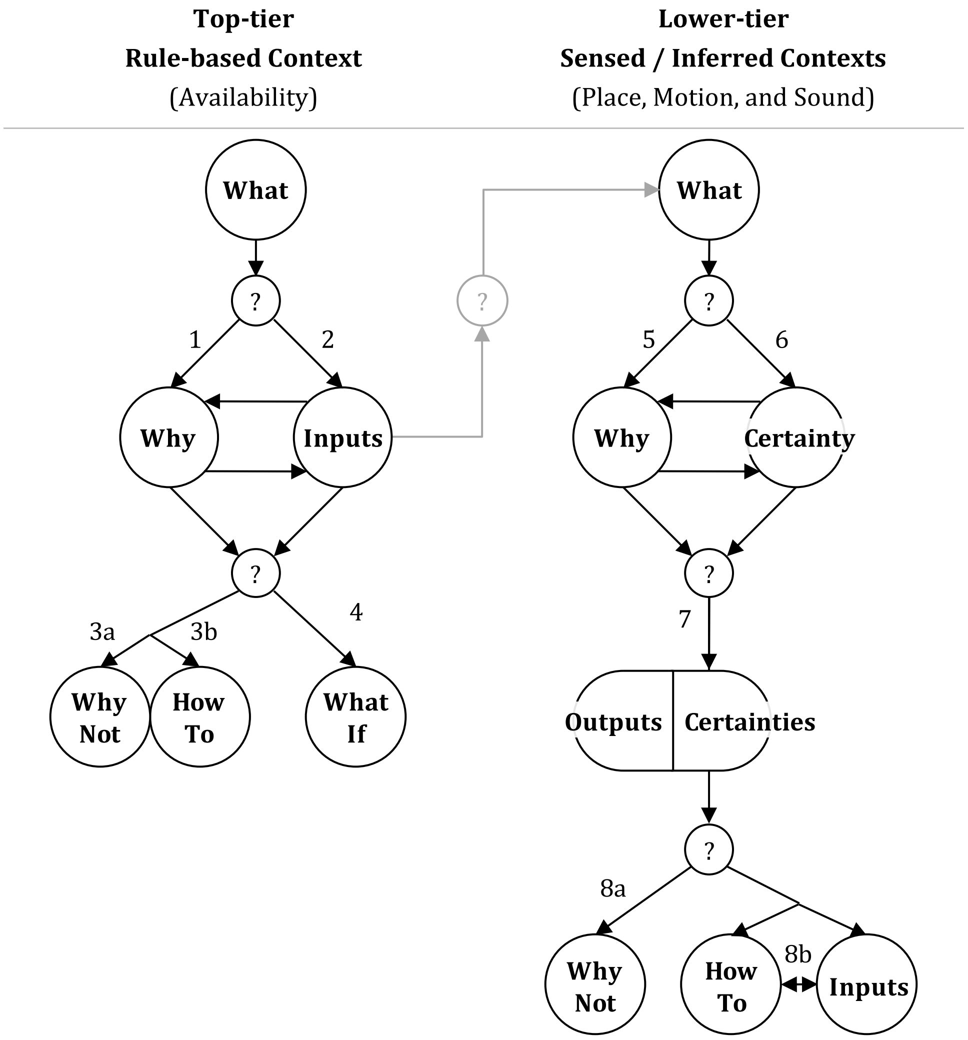


Figure 7.4. Consolidated sequence models of explanation use for Fault Finding. Steps were abstracted from individual actions that participants took across scenarios S3, S4, and S6.

7.8.3.1 TOP-TIER, RULE-BASED AVAILABILITY

When participants realized that the Availability was wrong, some *instinctively* first selected the Why question (Figure 7.4: 1). P6 first asked why about her Availability in S3 and S4, and why her Place was inferred as Office in S4. When explaining her rationale to first select why (S4), P12 said "because I want to know *why*…*why* I'm available." This suggests a *linguistic cue* for asking Why first.

Alternatively, sometimes participants first *inspected* the state of the application by asking for Input values (2). For S3, by examining the Input values, P11 discovered that "the Talking and possibly the Work [category] in the schedule [were] the two that led my status to be unavailable." For S4, P6 and

P12 quickly discovered that the Place inference (as Office) was erroneous (should be Library instead).

If participants had an *expectation* of what the availability should be, they would ask about the expected outcome using Why Not or How To questions (3a, 3b). These represent different strategies to address this goal-oriented query. Interestingly, some participants asked How To instead of Why Not, even though the latter was more *concise*; *e.g.*, for S4, P7 asked When would status be Semi-Available (How To) to *manually* identify erroneous conditions out of three. Had she asked "Why isn't status Semi-Available", she would have seen the single condition that was specifically identified to be relevant to the scenario. Unfortunately, some participants *misunderstood* the How To explanation, *e.g.*, for S7, when P4 asked "When would Sound be Unavailable," he interpreted the requirement that his friend has to be Talking to represent that his friend was currently sensed as talking; for S4, P7 examined the rules for when she would be Semi-Available (as she had expected status to be), found the condition Place = Library, and misunderstood that to mean that Laκsa had correctly inferred her location. Another reason for the lack of use of Why Not could be that the explanations with *contrapositives* put off users from using Why Not more: P6 complained about the "excessive use of negatives."

Some participants *simulated* conditions they expected to be true with the What If explanation (4), to see if Laκsa would infer an expected Availability. For S4, P8 asked What If, setting Place to Library (which she believed to be ground truth), and Ringer to Silent (which she believed she should have set). This resulted in the status of Semi-Available, which was correct unlike the actual inference of Available, and indicated that while the rule was executed correctly, possibly something was sensed wrongly. Unfortunately, participants were prone to *carelessness*: while setting up the expected state for S2, P6 changed three contexts (Place = Café, Motion = Sitting, Contacter = Friends), but failed to notice her schedule was set to Work (a pivotal factor to determine her status to friends). She expected her status to appear as Available, but it appeared as Unavailable instead; for S4, P13 forgot to set Place to Library (left as inferred value Office) when trying to verify the inferred availability.

Using the aforementioned strategies, some participants may decide to add a new rule or modify one to fix the anomaly, and this may be a satisfactory solution. However, most of the problems in the scenarios are due to faults at the lower-tier context inference. To investigate further, they would select the suspect context by clicking on the respective tab.

7.8.3.2 LOWER-TIER, SENSED / INFERRED CONTEXTS

Once again, participants *instinctively* asked Why (5): *e.g.*, for S2, P9 asked Why to see "that map thing," the Map visualization showing which place his location overlapped with; for S2, P6 asked Why Motion was inferred as Walking, only paying attention to which features were listed, and not paying attention to the boundary conditions; for S3, P6 first asked Why Sound was inferred as Talking, but found that unhelpful.

Participants also paid attention to the inference Certainty (6). For S6, after noting a Sound certainty of 73%, P10 mentioned that as long as it was above 50%, that was "good enough." For S7, P12 accepted the inference for Sound as Music (93% certainty), because it was above 90%. Subsequently, he was confused when this inference turned out to be wrong. When in doubt of the current inference, some participants also made it a point to find out which other Output values were *plausible* through their Certainties (7); *e.g.*, for S6, P6 checked the certainty of inferring Sound as Talking (8%), grew wary that the actual inference (Music at 73%) may be wrong, and became hesitant to contact his buddy. For S4, P12 wanted to see whether Laκsa recognized squatting, and seeing Certainties of 67% for Standing and 33% for Cycling, he accepted that "cycling is kind of like squatting" and "partially standing."

When asking about an *expected* outcome or exploring an *alternative* outcome with a noticeably high certainty, participants similarly demonstrated two dominant exploration strategies: asking Why Not (8a), or How To together with Inputs (8b). Asking Why Not provides a concise explanation that directly compares the actual inference with the desired outcome. For S4, immediately after noticing a wrong Place inference, P5, P6, P12 asked Why Not to see why their location bubble did not overlap with the bubble for Library. For S5, P5 used the Why Not Physical Diagram to explore how Running was not inferred (Cycling was). For S7, P6 became uncertain after noticing, from the Weights of Evidence visualization, that Pitch Fluctuation strongly voted for inferring that her friend was Talking, while every other factor voted for Listening to Music (see Figure 7.3d). However, we found that many participants disliked the explanations as being *too technical*, particularly, the Physical Diagrams for Motion. In fact, for S5, focused on Cycling, P7 avoided the Motion diagrams of the Input, Why, and Why Not explanations. For S6, P9 found the Weights of Evidence visualization explaining "Why isn't (Why Not) Sound Ambient Noise" confusing (difficult to remember what icons meant), and preferred to just look at the Output Certainty of Ambient Noise. Other participants alternatively used the How To explanation in conjunction with Inputs, by *manually* comparing the two explanations; *e.g.*, for S6, P9 repeatedly toggled between the How To and Inputs metaphorical diagrams for Sound. He studied Pitch Pureness to see if the current Input value (=29) was "just about right" compared to the average value for Music (=34), and accepted the Sound inference of Music. After several exposures to both techniques, P12 realized (in S5) that the Why Not explanation for Motion provided similar information as How To + Inputs, and required less effort to inspect.

7.9 DISCUSSION — THEMES OF INTELLIGIBILITY USE

We created an affinity diagram of our coded findings to map out core issues and patterns of use. Here, we present the top three high-level themes of how, and why participants used or failed to use the intelligibility features.

7.9.1 INFORMATION OVERLOAD AND EXPLANATION DETAIL

While we intend explanations to express a comprehensive view of what Laκsa knows and how it infers, we run into the problem of *information overload*. Comprehensive explanations are too long and complicated for end-users. Even though we took several steps to reduce the explanation complexity, participants still complained about the remaining complexity. P1 pointed out (as expected) that there were too many questions to choose from, and she did not necessarily know which was best for her goals. P3, P7, P8, and P9 also complained about the large number of reasons provided for Availability explanations (up to 9), and the large number of Input features described for Motion and Sound. In fact, P3 suggested showing up to 3-4 reasons, most participants only paid attention to 1-3 features of Motion (especially just vigorousness and movement), and Sound (periods of silence, pitch range, pitch pureness). When trying to determine her friend's availability for S6, P10 grew tired of asking for explanations. She felt "like it was information overload," and that she "started getting less information about what he was doing." She started doubting whether her friend was actually listening to music or sleeping instead. Clearly, our lay users did not want a lot of explanation detail.

Furthermore, participants preferred Certainty explanations because of its single value (*e.g.*, P9). Similarly, P12 eventually showed a preference for the more concise Why Not instead of How To explanation for explaining Sound. While participants found the Motion and Sound Input feature details interesting, they also found them too technical for such a lay-user application (*e.g.*, P3, P7).

7.9.2 PRIOR KNOWLEDGE AND RELATABILITY

It is well-known that *prior knowledge* plays a role in learning and understanding, and we observed how that influenced how participants used and interpreted explanations. For example, since Laκsa uses access points to sense location, the geographical distribution of these points affects the inferred location, and the distance error. However, P4 did not know this, and had no idea in S4 that Place was wrongly inferred as not being the Library.

This problem was more widely manifest in a usability issue: some participants wanted to see explanations framed in terms *relatable* to their activity; *e.g.*, for S5, P1 wanted Motion vigorousness to be stated as “within the walking or running range” or in terms of exercise "high or low intensity". She found forces and orientation unhelpful to understand her exercise. While this can clearly help users in making sense of input values, this is less feasible if an input has multiple ranges of values for certain outcomes (*e.g.*, multiple ranges of orientation angles for sitting due to different resting angles of the phone in a pocket).

The participants' use of Inputs and How To explanations for a comparative method to derive a Why Not explanation also suggests this need to frame sensor features in terms of well-understood activities. For S7, to convince themselves that a sound heard was really talking, participants looked at the feature values in Inputs (representing the current state), and values typical of Talking (found by asking How To).

7.9.3 DIFFERENT STRATEGIES IN PROBLEM SOLVING

We observed that some participants employed *suboptimal* problem solving strategies to try to determine how Laκsa made inferences. We observed a lack of strategy and logical fallacies such as causal oversimplification. Even when provided with various explanation tools, some participants did not know how to effectively use them. For S2, P11 *sequentially* explored questions in the drop-down list of questions, while others (*e.g.*, P6, P12) chose the Why explanation *instinctively*. Many participants also asked How To to get a Why Not explanation. This required them to do *manual* work to identify which reason was relevant, when Why Not would have automatically selected it.

Participants also exhibited common logical fallacies. Many participants exhibited *causal oversimplification* [Damer, 2009], because as they looked at reasons (*e.g.*, from Why, Why Not), they mistakenly fixated on a single factor and ignored others. P6 and P11 felt that Schedule was the “explicit way” of saying whether they were available (S2). P4 thought that since his friend was at the office (S7), then he must be Unavailable, even though he also would have needed to be talking. Having formed this wrong belief that his friend must be busy, P4 persisted in looking for clues to verify his hypothesis, rather than revise it. He was thus unable to identify the problem without help from the experimenter.

7.9.4 BLAME SHIFTING

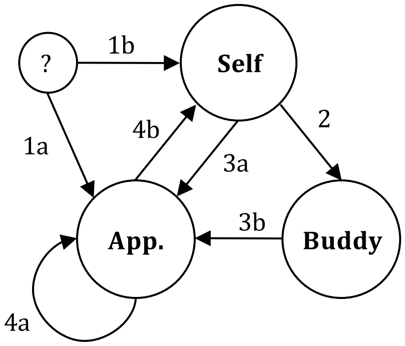


Figure 7.5: Blame shifting during S4 about mistakenly receiving a phone call in the library.

We observed participants *shifting blame* attribution in several scenarios, particularly, S4 where the participant received a loud call while at the library (see Figure 7.5). Participants changed who or what they attributed blame to after carefully considering what they knew, and viewing Laκsa's explanations. Immediately after receiving the interruption, P1, P7, P8, P12 reacted instinctively and were annoyed with Laκsa (1a), while P6, P13 were self-judgmental and felt they forgot to silence the ringer appropriately (1b). On reflection, P7, and P12 realized their coworker should have seen their Place and known not to call them. They would then blame their coworker for violating social norms (2). After looking at the availability status displayed, participants realized Laκsa was indeed wrong, and *all* participants shifted blame to Laκsa (3a, 3b). However, after viewing explanations and finding out that the status error was due to a poor location sensing (and both Library and Office in tight proximity), participants had different reactions. P8 and P12 continued to blame Laκsa for its imprecise sensing, and even became harsher in their judgment, because they (incorrectly) expected indoor location sensing to be as precise as contemporary GPS devices (4a). On the other hand, P6, P7, and P13 forgave it because they understood how challenging it was to sense location in the given circumstance, and understood that they would have to change a setting to improve sensitivity (4b). This supports findings about reduced blame attribution when autonomous robots explain their actions [Kim and Hinds, 2006]. While one might assume explanations improve perception and trust of Laκsa, this observation reveals explanations to be a double-edged sword: revealing the challenges of inference in some situations, or exposing the application's weaknesses.

7.10 DESIGN RECOMMENDATIONS FOR INTELLIGIBILITY

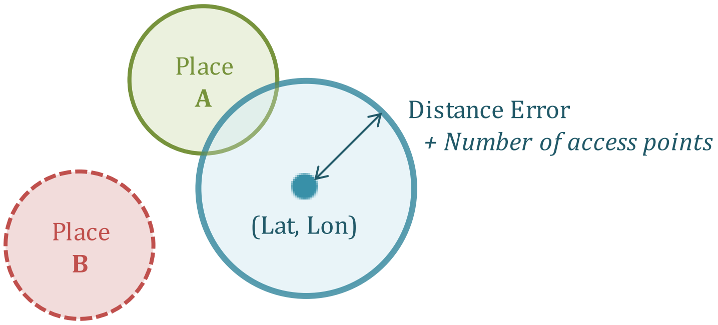
Drawing from our design exploration and user study, we present some recommendations on how to improve the usability of intelligibility that can be applied more generally to context-aware applications.

7.10.1 REDUCING AND AGGREGATING EXPLANATIONS

We had originally employed this strategy before the user study when designing Laκsa, and found this to be even more crucial based on our study results. In fact, participants demanded an even lower level of detail, wanting to see more details only as needed. Hence we continue to recommend this requirement. One compromise would be to allow incremental access to more detail *on demand* and offer significantly reduced explanations initially. Alternatively, it could be even better to present them in a form that is concise but does not compromise by omitting any reasons. Finding How To explanations cumbersome due to the large number of tabs to see multiple reasons, P1 suggested just presenting the rules in a table instead. This would provide a *bird's eye view* of the rules and yet be much easier to access. Furthermore, because some participants found some features for Motion and Sound to be overly technical, it may be sufficient to filter them out of explanations rather than make them physically meaningful, or provide metaphors to explain them. However, it is unclear whether users want to see them when encountering more serious and esoteric debugging problems.

7.10.2 RETOOLING EXPLANATIONS WITH SIMPLER COMPONENTS

Several participants (*e.g.*, P6, P9, P13) referred to explanations of Place as "the bubble thing" or "map thing" instead of noting which question they wanted to ask. They used the simple bubble components of Place (*e.g.*, see Figure 7.3a) to investigate various questions (Figure 7.6). Therefore, it may be better to design simple explanation components that can be used to answer multiple questions than to individually answer those questions through different automatically generated representations: *i.e.*, use explanation components with a smaller *vocabulary* set expressive enough to convey most of the explanation types we have employed. Unfortunately, it is difficult to design reusable, simple explanation components for contexts like motion and sound, because they depend on a wider and more diverse range of features that may not be represented equivalently.



What	Place inferred as at place A.
Certainty	Distance error and number of access points.
Inputs	Latitude and Longitude coordinates of sensed current user location.
Outputs	Place A, B, *etc*.
Why at A	Because sensed location bubble overlaps with bubble of A.
Why Not at B	Because sensed location bubble does not overlap with B.
How To be inferred at B	Need location and B bubbles to overlap. Certainty of inference:

Figure 7.6. Simple "bubble" components simultaneously for explaining seven questions about Place.

7.10.3 STREAMLINING QUESTIONING

Aware of her lack of understanding to effectively use the explanation questions, P7 suggested a *flow chart* to help guide users to ask optimal questions. We propose the *streamlined* flow of questions as shown in Figure 7.7, where only 1-3 question types are accessible at any given time. This helps reduce information overload when choosing explanations. Users first start with seeing What the application has inferred, along with its Certainty (1). If they want to ask questions, they can seek the mechanistic rationale by asking Why (2a), explore the system Inputs state (2b), or explore the Certainties of alternative Output values (2c). If users want to know why an expected or alternative output was not inferred, they may ask Why Not (3a), compare Inputs with How To (3b). These support the two observed why-not strategies. Having observed how participants wanted to ask questions from the Laksa UI, we recommend convenient shortcuts: users can ask Why Not on seeing alternative Output values (4), and simulate different Input values to ask What If (5). Finally, users can also explore the values (6a) and Outputs (6b) of lower-tier contexts through Inputs and What If, respectively.

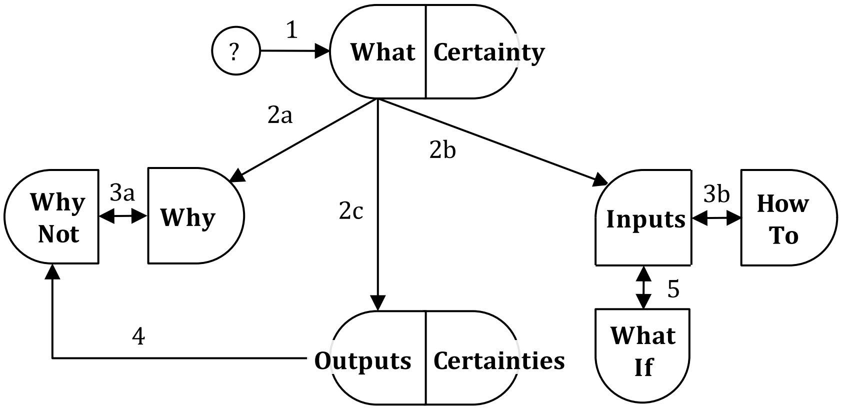


Figure 7.7. Streamlined sequence diagram for explanation use.

7.10.4 NON-MECHANISTIC EXPLANATIONS

We found that users need more types of explanations to properly ground them in the application domain, and to educate them about good problem solving and debugging strategies to fully understand the program functionality.

Given the deep knowledge that complex context-aware applications rely on to make decisions, and the evidence that some participants lacked sufficient prior knowledge to relate technical behavior to real-world and domain phenomena, we can see that simple textual descriptions are not sufficient to scaffold the automatically generated explanations. This suggests context-aware applications need to have access to information about complex real-world concepts that are not necessarily core to the application.

Moreover, we found that since some users did not effectively leverage the explanation facilities provided, intelligible applications may need to teach them problem solving strategies. One solution may be to provide examples of end-user debugging with the explanation tools.

7.11 LIMITATIONS AND FURTHER WORK

We used an iterative design process where design decisions were based on careful consideration, consultation with HCI experts, and user feedback. While we believe our designs are reasonably interpretable, an alternative approach is to make comparisons between competing designs.

We have limited our study to a manageable set of scenarios, chosen to aid exploration of explanation use, rather than to comprehensively cover situations. Hence, our results are suggestive rather than definitive, and we seek to validate them with further design iterations and user studies.

Our design recommendations are based on studying how and why users seek explanations given several goal situations. While we expect that following them would improve the usability and thus usage of explanations, they may not be the best to *promote understanding*. Future work will explore how to design and provide explanations that are not only easier to interpret, but also effective in improving the understanding of how context-aware applications work. In Chapter 9, we describe follow-up work on the intelligible Laĸsa prototype to explore the usage of intelligibility and how that affects end-user understanding of Laĸsa's context inference.

7.12 CONCLUSION

We have described our first steps to building a real-world, intelligible mobile context-aware application. We followed several design principles to improve the usability of explanations, and conducted a user study to discover how users make use of explanations, and issues they experienced. Particularly, we investigated the use of intelligibility for the mobile contexts of Availability, Place, Motion, and Sound activity. Our findings emphasize the importance of making explanations usable and quickly consumable (by reducing information overload), relating the application behavior to the real world activity (to raise the relevance of the information), and supporting effective problem solving and debugging strategies (so that users can quickly understand the application issues before giving up). We suggest a need for streamlining explanations while maintaining access to the rich explanation capabilities, and for integrating domain knowledge in explanations. With a better understanding of how users use the question type explanations, we can better design explanations to help understand sophisticated context-aware applications.

8 Evaluating Intelligibility Under Uncertainty

This chapter is an extension of the work presented in:

> Lim, B. Y. and Dey, A. K. (2011). Investigating Intelligibility for Uncertain Context-Aware Applications. In *Proceedings of the 13th international conference on Ubiquitous computing (UbiComp '11)*. ACM, New York, NY, USA, 415-424.

Abstract. Context-aware applications use sensing and inference to attempt to determine users' contexts, and take appropriate action. However, they are prone to uncertainty, and this may compromise the trust users have in them. Providing intelligibility has been proposed to help explain to users how context-aware applications work in order to improve user impressions of them. However, we hypothesize that intelligibility may actually be harmful for applications that are very uncertain of their actions. We conducted a large controlled study of a location-aware and a sound-aware application, investigating the impact of intelligibility on understanding, and user impression of applications with varying certainty. We found that intelligibility impacts user impressions, depending on the application's certainty and behavior appropriateness. Intelligibility is helpful for applications with high certainty, but it is harmful if applications behave appropriately, yet display low certainty.

8.1 Introduction

In previous chapters, we have found that some explanation types were more effective than others in improving understanding and trust (Chapter 4), and later investigated more explanation types that end-users of context-aware applications are interested in (Chapter 5). However, even though these studies show great promise for the efficacy of intelligibility in context-aware applications, they have assumed the use of systems that have reasonably high certainty in their actions, and that, while fallible, generally take appropriate actions. Intelligibility would enhance the positive impression a user may have of an application, and reveal how it intelligently tries to figure out what is happening even for difficult sensing and inference situations. Unfortunately, because of these difficulties in sensing and inference, applications can be uncertain of their actions, often resulting in users having a negative impression of these applications. It is hoped that intelligibility would help bring up this shortfall, and raise a user's impression of a context-aware application. However, is there a certainty below which intelligibility would not help, but may actually *harm* a user's impression of the application? If this were the case, the user could lose even more trust in the application's capability and precision. So an application with sufficiently low certainty would not benefit from adding intelligibility, and instead, the developer should focus on improving its certainty instead.

In this chapter, we present two scenario-driven lab studies where we investigate the interaction between intelligibility and application uncertainty. For the first study, we manipulated the provision of Intelligibility in three levels (None, Certainty-only, Full), and Certainty in six levels (50, 60, 70, 80, 90, 100%), in a *between-subject* design for an online survey. We designed two context-aware applications (location-aware, and sound-aware) to explore the impact of certainty on intelligibility for applications with differing complexity. In a follow-up study, we ran a think-aloud study using a reduced form of the online survey, seeking to add greater context to our quantitative findings. Our contributions are:

1. Understanding how users respond to intelligibility in context-aware applications under different levels of certainty; and
2. Identifying when, how, and why intelligibility is helpful or harmful as a result of application certainty.

8.2 INTELLIGIBILITY AND UNCERTAINTY

In this section, we provide background and related work on intelligibility, uncertainty in context-aware applications, and the impact of showing uncertainty to end-users.

8.2.1 DISPLAYING UNCERTAINTY IN CONTEXT-AWARE APPLICATIONS

Context-aware applications are prone to uncertainty, and one common strategy for dealing with this involves user mediation where the user resolves uncertainty [Dey *et al*., 2002]. Furthermore, these applications should represent to their users what they know [Bellotti and Edwards, 2001], not hide this ambiguity or uncertainty [Greenberg, 2001], and reveal the "seams" of their underlying systems [Chalmers and MacColl, 2003]. Currently, some context-aware applications are able to model uncertainty due to their underlying probabilistic models (*e.g.*, [Kulesza *et al*., 2009; Tullio *et al*., 2007]), but few display the system certainty (*e.g.*, [Cheverst *et al*., 2005]).

Conversely, many studies have also explored various ways to display uncertainty, and the benefits of doing so. Antifakos and colleagues showed that uncertainty improved task performance speed of participants when certainty is high [2004], and that participants verified automatic settings made by a context-aware system less often when its certainty was high or medium [2005]. Similarly, Rukzio *et al*. [2006] found that displaying uncertainty slowed down user performance, because users would double-check fields with lower certainty. In studies of presenting location information, visualizations of location certainty were found to improve user performance with location-based services [Dearman *et al*., 2007; Lemelson *et al*., 2008]. Though not explicitly investigating about uncertainty, Yan *et al*. [2010] found that displaying higher trust and reputation values of mobile applications increased users' willingness to continue using them.

Our work adds to the research on displaying uncertainty by carefully varying uncertainty to identify a certainty threshold below which displaying uncertainty becomes harmful instead of helpful, in two different contexts — location and sound. Furthermore, we extend the displaying of uncertainty to include other explanations that provide users with a fuller form of intelligibility.

8.2.2 INTELLIGIBILITY IN CONTEXT-AWARE APPLICATIONS

For this work, we use the definition of intelligibility defined Chapter 5, which classifies explanations in terms of questions that users may ask of context-aware applications. Specifically, we developed interfaces for explanations of the following questions:

1. **What** is the current value of the context?
2. **Certainty**: how certain is the application of this value?
3. **Why** is this context the current value?
4. **Why Not**: why isn’t this context value Y, instead?
5. **Inputs**: what factors affect this context?

We describe how to provide explanations for these questions later in Section 8.5 (Application Platforms).

8.3 HYPOTHESES

While we do not assert it here, we believe the user's impression of an application impacts her trust of it. We define that a user has a good *impression* of a context-aware application when she perceives it to be highly **certain** of its inference, feels that it generally behaves **appropriately**, and she **agrees** with what it is doing. As illustrated in Figure 8.1, we hypothesize that:

H1a: Above a certainty threshold, intelligibility improves a user's impression of a context-aware application.

H1b: Below the threshold, intelligibility harms the user's impression of the application. This could be due to the user realizing how poorly the application is performing.

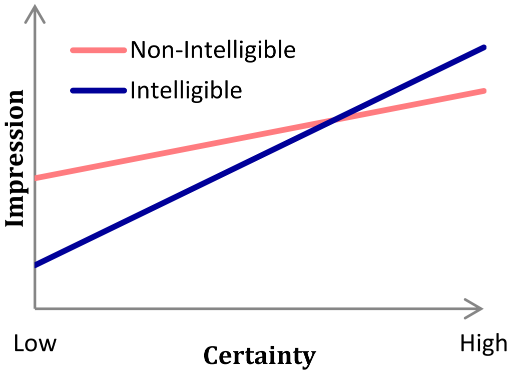


Figure 8.1. Hypothesis 1: Intelligibility will improve user impressions when an application is certain of its actions, but it will harm impressions when it is uncertain. Only interaction effect suggested, not linearity of trends.

We hypothesize that this effect on impression is due to the increased understanding provided by intelligibility:

H2: Providing intelligibility helps increase a user's understanding of the application.

While H2 has been shown to be true in Chapter 4 [Lim and Dey 2009], we seek to verify those results, as H1 depends on this. Thus, H2 in combination with H1b hypothesizes that a *gain* of understanding about a low certainty application leads to a *loss* in impression. Next, we describe a large-scale, between-subjects lab study to test these hypotheses.

8.4 METHOD

We are primarily interested in the interaction between the provision of intelligibility, the certainty of the application, and the impact on understanding and impression. We chose to investigate this effect using a large-scale, controlled lab study. The study was deployed online through Amazon Mechanical Turk (MTurk) to allow us to collect input from a large number of participants and span many levels of certainty and intelligibility. For generality, we designed two context-aware applications and varied their certainty, and intelligibility levels. We exposed participants to several canonical situations of these applications through 10 different scenarios.

8.4.1 EXPERIMENTAL CONDITIONS

We varied Intelligibility and application Certainty as independent variables in a *between-subject* experiment, across two applications, for a total of 3×6×2=36 conditions.

8.4.1.1 INTELLIGIBILITY (3 CONDITIONS: NONE, CERTAINTY-ONLY, FULL)

We varied whether participants were provided with explanations where they only saw the application inference (None), or additionally saw a rich explanation visualization (Full). We included an intermediate intelligibility level, where we provided just Certainty percentage only, to investigate how much value the explanation visualizations add over just showing certainty.

8.4.1.2 CERTAINTY (6 CONDITIONS: 50%, 60%, 70%, 80%, 90%, 100%)

We varied certainty as six intervals (rather than a dichotomy) to be able to observe any trends that may arise.

8.4.2 MEASURES

We are interested in measuring how much participants understand the application for each intelligibility condition, and whether this affects their perception of certainty, feeling of whether the application behaved appropriately, and how much they agree with the application's inference.

Understanding. For each scenario, we asked participants *why* the application inferred what it did, and *why not* something else (free-text). We asked these questions for all scenarios to prime participants to think about the underlying inference of the application. We analyzed the responses from the sixth of 10 scenarios presented, as we expected participants to be sufficiently familiarized with the application through previous scenarios, but not overly tired of providing feedback. We validated that this was true through a sampling of the responses.

Perceived Certainty. For each scenario, we asked participants how certain they believed the application was in its inference (as numerical input 0 to 100%). After the scenarios, we asked for their overall sense of the certainty.

Perceived Appropriateness. For each scenario, we measured what the participant felt about the appropriateness of the application behavior, on a 7-point Likert scale from Very Inappropriate to Very Appropriate.

Agreement. For each scenario, we measured how much the participant agreed with the application's inference, given the ease or difficulty of making the inference; on a 7-point Likert scale from Strongly Disagree to Strongly Agree.

8.5 APPLICATION PLATFORMS

To investigate the interaction between intelligibility and uncertainty, we designed two applications — LocateMe and HearMe — and varied their certainty and intelligibility levels. Derived from design explorations of intelligibility in Chapter 7 [Lim and Dey, 2011a], both applications are mobile phone applications, but deal with different contexts (location, and sound activity, respectively), different inference mechanisms, and different explanation interfaces. While real, physical prototypes were not used in this study, these applications have been prototyped, and their described functionality are feasible and indicative of real applications and their associated uncertainty. We describe these applications, how they sense and make inferences, their basis for uncertainty, and how they visualize their inferences.

Each application has three different levels of intelligibility. The None version would just show the output of the application (*e.g.*, "You are at the Washroom", "You were in a Conversation"). The Certainty version adds a certainty percentage (*e.g.*, 89%, 62%). The Full version adds an explanation visualization (see Table 8.1 and Figure 8.4).

8.5.1 LOCATEME

LocateMe is a location-aware mobile phone application that uses GPS, Wi-Fi and cellular networks to triangulate where the user is, and match that to a predetermined set of locations to infer which *place* the user is at. It then uses this inference to take actions such as sending a reminder, or identifying the nearest printer. In the scenarios, LocateMe is used for indoor and outdoor situations.

Basis for uncertainty. Due to the probabilistic model of the user's location (as a Gaussian area), LocateMe infers the user being at places with varying levels of certainty. Its certainty depends on how much the user's estimated area "overlaps" with the area of the named place, and is computed into a probability. The larger the area of the named place, and/or the closer the user's area is to that place, the higher the certainty. Uncertainty is also affected by sensing errors due to GPS signal occlusion (*e.g.*, being indoors), Wi-Fi or Cell network signal strength, *etc*.

8.5.2 HEARME

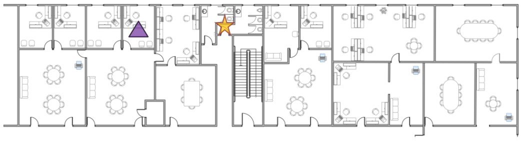
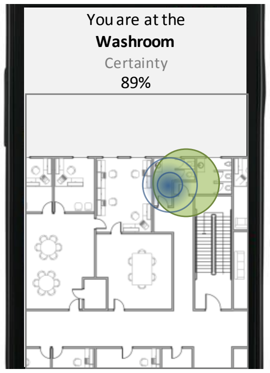
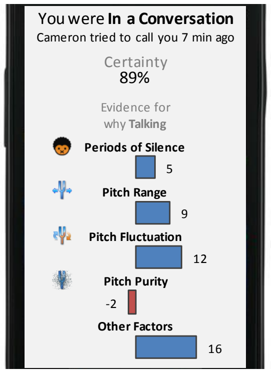
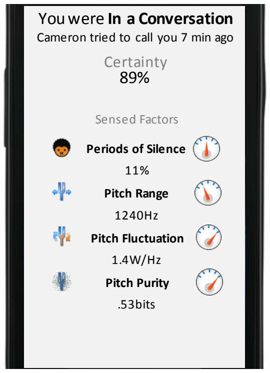
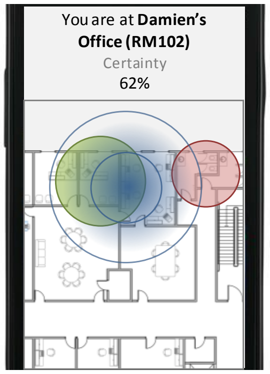
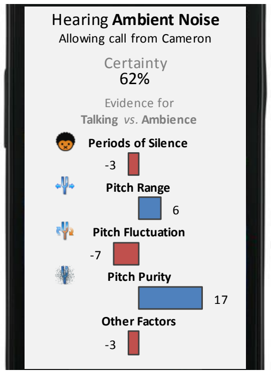
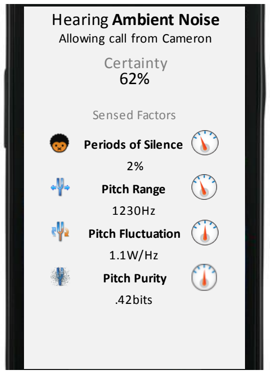
HearMe is a sound-aware mobile phone application that uses the phone's microphone to sense and infer one of three activities: whether the user is (i) in a conversation, (ii) listening to music, or there is mostly (iii) ambient noise. It uses several features extracted from processing the microphone signal, such as: frequency bandwidth, spectral entropy, low-energy frame rate, Mel-Frequency Ceptral Coefficients (see Section 7.2 for more details about features used). HearMe uses a trained naïve Bayes model to infer whether the sound heard was one of the three activities.

Basis for uncertainty. HearMe models uncertainty of its inference from the probabilistic uncertainty of the naïve Bayes model. This depends on the sound samples used to train the original model. HearMe does not model the error due to the microphone signal for uncertainty.

Due to the ubiquity of GPS devices and location sensing in smart phones, LocateMe is likely more familiar to users than HearMe which uses machine learning inferences that are less common-place in devices available to consumers.

8.6 SCENARIOS

Similar to [Antifakos *et al*., 2004; Lim and Dey, 2009], we use scenarios to let participants learn about and experience our applications. However, rather than present 5-second video clips to help participants experience a scenario, we provided users with a precise representation to understand the ground truth of each scenario. For LocateMe, we showed a map or floorplan indicating where the participant would actually be in the scenario. For HearMe, we played an audio clip of what the participant and her phone would supposedly have heard.

	LocateMe	HearMe
Situation description and Ground truth	**(a)** You are in the washroom taking care of some business, just before your meeting with your neighboring coworker, Damien.  Star denotes where you actually are at; purple triangle denotes Damien's office (RM102).	**(d)** At the coffee shop, you find Michelle, a coworker, there, and have a chat with her. *Participant listens to an auto-started audio clip of ambient noise in a coffee shop, with a female voice occasionally talking.* H6-groundtruth.mp3 ▶
Application behavior **Certainty 90%**	**(b)** You receive a text message from Damien, who tells you he is waiting for you at his office. You check LocateMe:  You are at the **Washroom** Certainty 89%	**(e)** You are not interrupted for 12 min, and when the conversation ends, you receive a notification message from HearMe:  You were **In a Conversation** Cameron tried to call you 7 min ago Certainty 89% Evidence for why **Talking** **Periods of Silence** 5 **Pitch Range** 9 **Pitch Fluctuation** 12 **Pitch Purity** -2 **Other Factors** 16  You were **In a Conversation** Cameron tried to call you 7 min ago Certainty 89% Sensed Factors **Periods of Silence** 11% **Pitch Range** 1240Hz **Pitch Fluctuation** 1.4W/Hz **Pitch Purity** .53bits You see that Cameron had tried to call you, but HearMe suppressed his call since it interpreted you as uninterruptible.
Certainty 60% ***Wrong inference, in this case***	**(c)** Damien calls to ask where you are since LocateMe said you are in his office, which is obviously false. You check LocateMe:  You are at **Damien's Office (RM102)** Certainty 62%	**(f)** You are not interrupted for 12 min, and when the conversation ends, you receive a notification message from HearMe:  **Hearing Ambient Noise** Allowing call from Cameron Certainty 62% Evidence for **Talking** *vs.* **Ambience** **Periods of Silence** -3 **Pitch Range** 6 **Pitch Fluctuation** -7 **Pitch Purity** 17 **Other Factors** -3  **Hearing Ambient Noise** Allowing call from Cameron Certainty 62% Sensed Factors **Periods of Silence** 2% **Pitch Range** 1230Hz **Pitch Fluctuation** 1.1W/Hz **Pitch Purity** .42bits

	LocateMe	HearMe
Explanation UI Description	LocateMe uses "bubbles", to determine and show where the user is, rather than showing pin-point positions. The user's sensed location is represented by two blue concentric circles, and a Gaussian blue area. He is most likely to be in the center of the area, but less likely the further away from it. The bounds indicate thresholds of certainty (50, 90%) that the user is within the bounds. Places are represented with uniform circular areas of varying size. *E.g.*, Washroom is defined with a circle, with the center where the room is, and the size is how large the room is. The user is inferred to be at a place if his blue bubble "overlaps" with the place's bubble. A green bubble indicates the place where the user is inferred to be; a red bubble indicates where he is not inferred to be. (b) shows a green bubble over the washroom overlapping with the user's blue bubble to explain *why* he is inferred to be there. The large overlap suggests a high certainty (in this case, 89%). (c) explains *why* the user is *not* inferred to be at the washroom but at Damien's office instead, by showing: a red bubble for the washroom, a green bubble for Damien's Office. The blue bubble overlaps with the green bubble more than with the red bubble, indicating 62% certainty.	HearMe uses two types of visualizations to explain what it senses, and how it infers an activity, with a Sensed State and Evidence visualization, respectively. We substitute the technical names of the input factors with metaphorical terms (*e.g.*, Periods of Silence for low-energy frame rate, Pitch Purity for spectral entropy), and aggregate the remaining factors as Other Factors. We explain the meanings of each factor and implications of their values, *e.g.*: Periods of Silence indicates what percentage of the sound sample was relatively silent compared to the rest of it; talking would have higher percentage. The Sensed Factors viz (right diagram) shows the values of the factors, and a gauge icon indicating whether each value is at, below, or above the average values for that factor. The Evidence visualization (left diagram) shows a bar chart indicating if each factor votes for (blue towards right) or against (red towards left) the inference, and by how much. This viz can be used to compare one output against all others (e), or specifically contrast between two outcomes (f). The balance of the bars indicate how certain the application is about its inference. If it is more certain, the bars are weighted more towards the right, and if less certain, the bars are equally weighted to the right and left.

Table 8.1. Scenario scripts, application interfaces showing Full intelligibility, and their interpretation of Scenario 6.

We presented 10 scenarios as a chronological sequence of events happening through a single day. As in Section 7.6.2 [Lim and Dey, 2011a], the scenarios were written to span five *themes* typical of what context-aware applications are used for: interruption management, social awareness, reminders, recommender, exploration / learning. Each theme is *repeated* twice (not consecutively) to provide repeated exposure. Collectively, the scenarios are *representative* of the application certainty (*e.g.*, for 60%, the application behaved appropriately for 6 out of 10 scenarios). Hence, participants in the None intelligibility condition could perceive the certainty of the application. The certainties presented (for Certainty and Full intelligibility) also reflected the certainty condition, but with small *randomized* differences (*e.g.*, 60, 63, 59, 60, 58, 62, 61, 57, 60, 60%), to prevent participants from ignoring the values had they been constantly shown 60% repeatedly.

Table 8.1 shows the scripts and diagrams shown to participants in the LocateMe (left), and HearMe (right) surveys for a scenario, Table 8.2 describes how different explanation types are provided in both applications. S6. Next, we describe what participants were asked to do for each application survey.

	Description / Function	**LocateMe**	**HearMe**
What	Show the current inference of the context and consequent action.	Reports inferred place, and shows blue bubble of the user in map visualization.	Reports inferred sound activity.
Certainty	Show the certainty of the application's inference of the current context value.	Shows a certainty percentage, and size of bubbles in map visualization (larger sizes show lower certainty).	Shows a certainty percentage, and sense of balance of bars in evidence visualization. (more balance show lower certainty).
Why	Show a model-based explanation of how the application inferred the current context value.	Shows the overlap between the inferred place (as a green bubble) and the user's blue bubble.	Shows weights of evidence for the inference due to each input factor in a bar chart visualization.
Why Not	Show a model-based explanation distinguishing how the application did not infer the alternative inference.	In addition to the Why visualization, shows the lack of overlap between the place (as a red bubble) and blue bubble.	Shows the evidence visualization, contrasting the current inference against the alternative inference.
Inputs	Show the current values of input context / features.	Visually shows the user's position by positioning the blue bubble in a map.	Lists current input factor values, and provides a gauge of its relative value.

Table 8.2. Explanation types. LocateMe uses a map and bubbles visualization for its explanations about its location inference. HearMe uses lists the current values of its sensed factors, and their corresponding evidence to explain its sound activity inference.

8.7 PROCEDURE

After consenting to participate in the survey (either LocateMe or HearMe), the participant was randomly assigned to a Certainty condition and an Intelligibility condition. He read instructions on how the application works, and how to interpret its display. As recommended by [Kittur *et al.*, 2008], we then asked two verification questions (multiple-choice) to ensure comprehension. The participant next went through 10 scenarios to experience the application under various situations.

For each scenario, he read (i) a scenario description, and (ii) the subsequent response of the application which may or may not be appropriate for the situation. He was then (iii) asked verification questions to ensure he had carefully read and understood the scenario. Next we asked questions for our measures of (iv) perception of certainty, (v) application behavior appropriateness, and (vi) agreement. Finally, he was asked about his (vii) understanding of the application inference. After the scenarios, the user was tested on his (viii) overall understanding of the application and (ix) overall perceived certainty. Finally, he was asked about his background with using smart phones, and for demographic information.

8.8 PARTICIPANTS AND DATA CLEANSING

We recruited participants from Amazon Mechanical Turk. There were 584 completed HITs (human intelligence tasks), and 397 incomplete HITs. We rejected 76 HITs because each participant had low verification score, rushed through the survey too quickly, and/or was unconscientious (gave reasons that were gibberish, repetitive, or irrelevant). Of the remaining 508 participants, their survey completion time was Median=33 minutes (8.9 to 109), and their verification score was Median=20 (7 to 22) out of 22. Some participants had low verification scores, which indicates poor understanding of the scenarios and application, but their free-text reasons indicated conscientious effort in the survey. So they were included in our population sample to represent users who have greater comprehension difficulty. We had participants across 36 conditions (3 Intelligibility × 6 Certainty × 2 Application) in our experiment (M=14.1, 11 to 17 in each condition). We paid each participant $2.

8.9 DATA ANALYSIS AND RESULTS

In this section, we present the analysis we performed on the survey results, related to our hypotheses. Before we investigate whether intelligibility influences users' impressions of a context-aware application, first we analyze whether intelligibility improves understanding of how the application works (H2). We assume that understanding is not influenced by the certainty of the application.

8.9.1 UNDERSTANDING OF APPLICATION INFERENCE

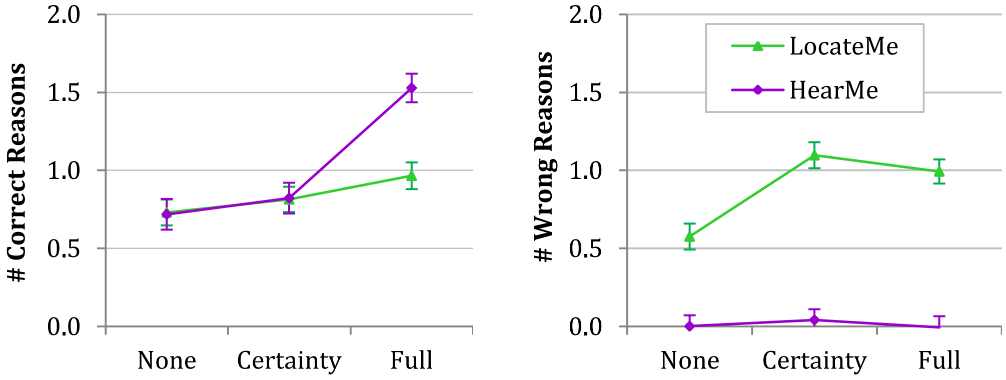


Figure 8.2. Mean of number of correct and wrong reasons counted from participant free-text responses of how the application made its inference in S6. Participants with Full intelligibility gave more correct reasons when explaining about HearMe than those with None (p<.01); this was only marginal for participants explaining LocateMe (p=.08). Furthermore, participants explaining LocateMe offered more wrong reasons than those explaining HearMe (p<.01), particularly when provided with some form of intelligibility (p<.01).

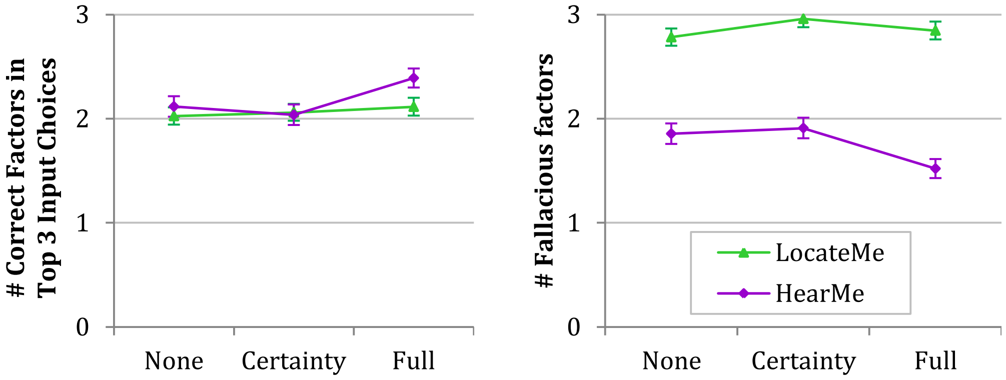


Figure 8.3. Mean number of correct Input factors in the top three choices of the Inputs ranking task (Left), and number of fallacious factors given in all 10 choices (Right). Participants with Full intelligibility chose more correct factors of HearMe as top three than those with None (p=.014); this was not noticeable for LocateMe (p=n.s.). Moreover, participants explaining HearMe chose fewer fallacious factors than those explaining LocateMe, particularly when provided with Full intelligibility (p<.01).

As a measure of understanding, we coded the free-text responses about how they thought the application made its inferences for S6. We counted how many of the reasons *about the application* that they provided were correct. A reason is considered correct if it relates to an actual factor that the application uses (*e.g.*, GPS, latitude, distance threshold, bubbles; Periods of Silence, Pitch Purity, noisiness). We eliminated repeated and redundant reasons (*i.e.*, paraphrasing of the same idea), and accepted language that demonstrated an approximate idea of valid concepts.

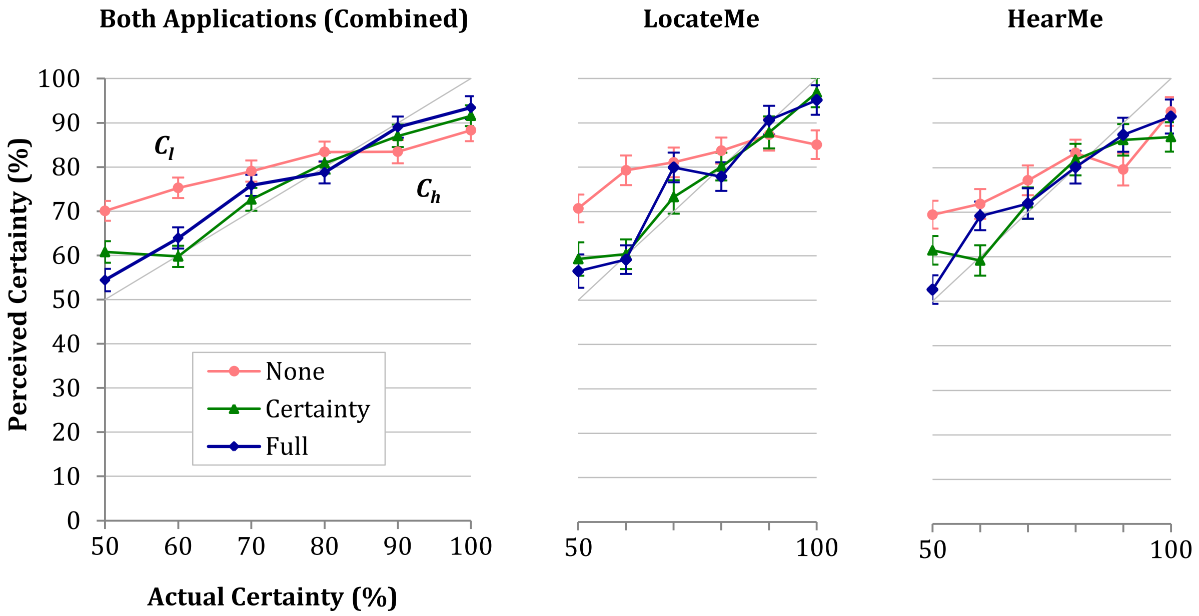
We fit a mixed model with: correct reason count as the dependent variable, intelligibility and application as independent variables, interaction × application as an interaction effect, certainty as a control variable, and participant as a random variable (nested in intelligibility, application, and certainty). We found that participants with Full intelligibility gave more correct reasons than those with None, especially regarding HearMe (see Figure 8.2 and Figure 8.3).

Next, we analyze whether and how this increase in understanding influences how participants perceived the certainty of the application. We annotate some figures to indicate notable findings in our results (*e.g.*, C_l, $C_{\neg a,l}$, A_a).

8.9.2 Perceived Overall Certainty

We asked each participant about their perception of the overall (average) certainty of the application, after completing all 10 scenarios. To examine differences in this perception for each application separately, we fit a mixed model with: perceived overall certainty as dependent variable, intelligibility and certainty as independent variables, intelligibility × certainty as an interaction effect, and participant as a random variable (nested in intelligibility and certainty). We also combined data from both applications, and fit a similar mixed model but also with application as a control variable. These results are presented in Figure 8.4 and Table 8.3.

Our results show that, for high actual certainty, participants with intelligibility perceived a higher certainty than those without (C_h); for low actual certainty, participants with intelligibility perceived a lower certainty than those without (C_l). Alternatively, an interpretation may be participants with intelligibility just copied the certainty displayed. Means testing suggests that this could be so (see Table 8.4), but we further investigate this in a follow-up study (see later).



F(10,489)=4.02, p<.001 F(10,489)=3.77, p<.001 F(10,489)=1.97, p<.05

Figure 8.4. Perception of Overall Certainty: combined analysis (Left), and for individual applications (Middle and Right). Participants with Full intelligibility perceived a higher certainty when the application had high actual certainty, but perceived a lower certainty when it had low actual certainty.

Application	Combined		LocateMe		HearMe	
Actual Certainty	**Low** 50-70%	**High** 90-100%	**Low** 50-60%	**High** 100%	**Low** 50-70%	**High** 80-100%
None *vs.* Full	p<.001	p<.05	p<.001	p<.05	p<.01	p=n.s.
None *vs.* Certainty	p<.001	p=n.s.	p<.001	p<.05	p<.01	p=n.s.

Table 8.3. Pre-hoc contrast between Intelligibility types for low and high actual certainty. These groups were chosen after visually inspecting the interaction graphs.

Certainty (%)	50	60	70	80	90	100
Certainty	<.01	n.s.	n.s.	n.s.	.05	.01
Full	.02	n.s.	<.01	n.s.	n.s.	<.01

Table 8.4. Means testing of whether perceptions of overall certainty are different from actual certainties. t-test p-values suggest copying if p=n.s.

While our results show that participants' perceived overall certainty *across* different certainty levels is influenced by intelligibility, we next show that their perception also varies based on how the application behaved per scenario.

8.9.3 PERCEIVED CERTAINTY BY APPLICATION APPROPRIATENESS

To investigate perception of certainty across scenarios, we analyzed the repeated measure of how certain participants felt the application was for each scenario. Figure 8.5 (Right) shows the fluctuation of perceived certainty as participants with no intelligibility (None) go through the scenarios, depending on whether the application behaved appropriately in the scenario. When participants received Full intelligibility (Figure 8.5, Left), their perceived certainty was more stratified, and less fluctuating.

We group our results by application appropriateness, and fit two mixed models with: perceived certainty as dependent variable, intelligibility and certainty as independent variables, intelligibility × certainty as an interaction effect, and participant as a random variable (nested in intelligibility and certainty). Our results (see Figure 8.6 and Figure 8.7) show that, for appropriate application behaviors, participants with intelligibility perceived lower certainty than those with None when encountering actual low certainty ($C_{a,l}$), and conversely perceived higher certainty when encountering actual high certainty ($C_{a,h}$). For inappropriate application behaviors, there was no difference in perception for actual low certainty ($C_{\neg a,l}$), but participants with intelligibility perceived higher certainty for actual high certainty, than participants without ($C_{\neg a,h}$).

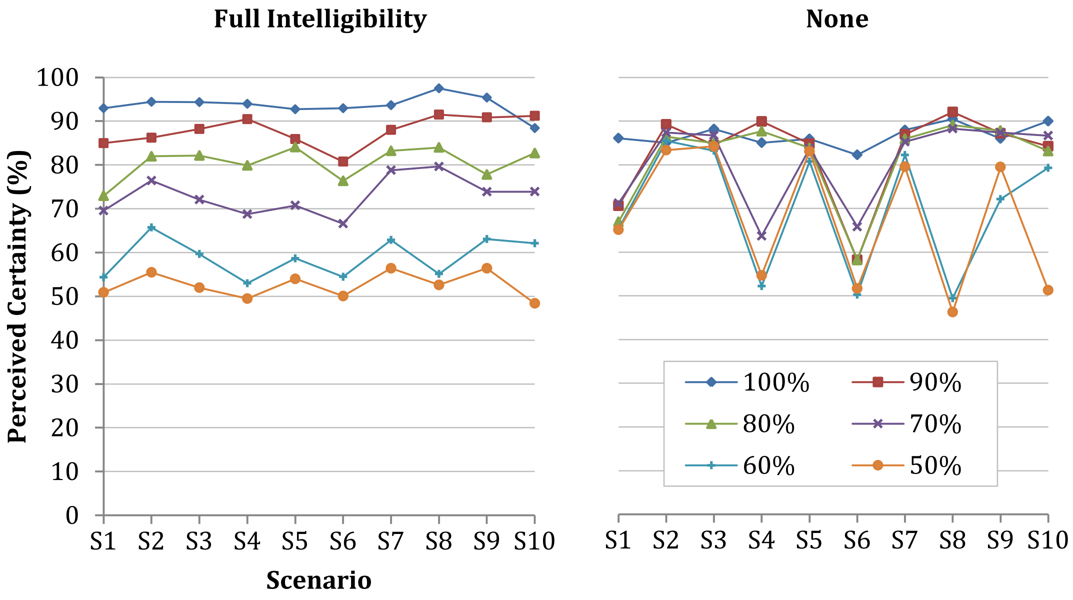


Figure 8.5. Perceived certainty influenced by application appropriateness across scenarios. The application behaved appropriately for at least one Certainty condition in S1, S4, S6, S8, and S10, more so for lower certainty conditions.

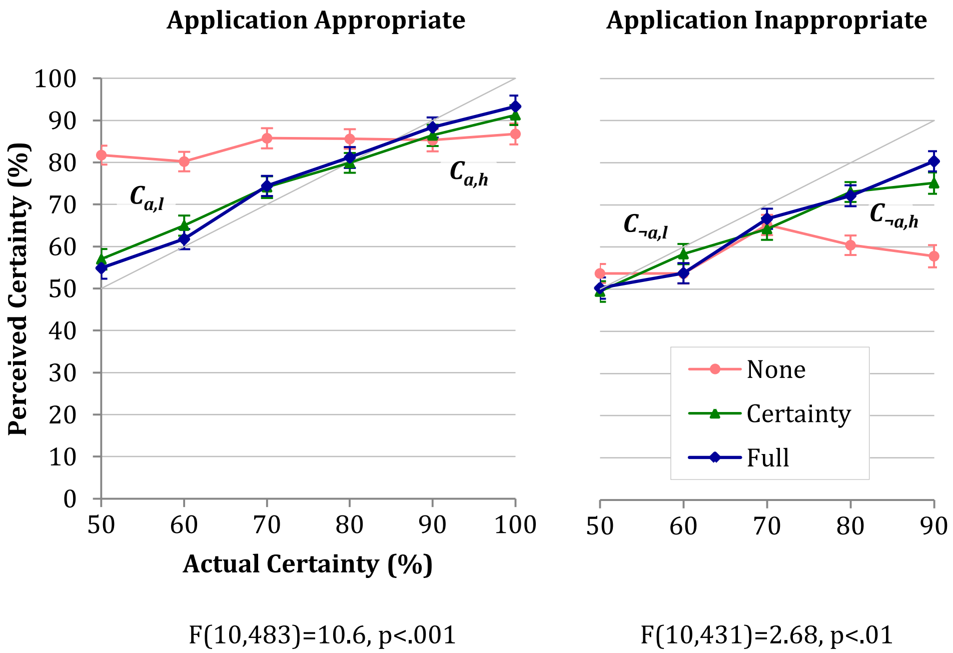


Figure 8.6. Perceived certainty across actual certainty by application appropriateness. Note: no inappropriate scenarios for 100% certainty condition.

App Behavior	**Appropriate**		**Inappropriate**	
Actual Certainty	**50-70%**	**80-90%**	**50-70%**	**80-90%**
None *vs.* Full	p<.001	p<.05	p=n.s.	p<.001
None *vs.* Certainty	p<.001	p=n.s.	p=n.s.	p<.001

Table 8.5. Contrast between Intelligibility types for low and high actual certainty grouped by application behavior.

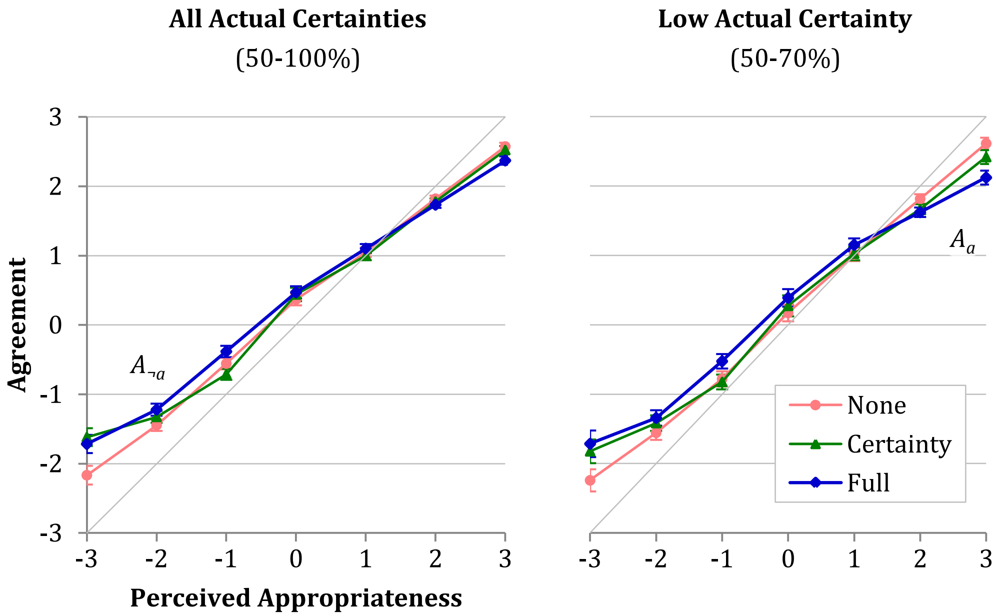
8.9.4 AGREEMENT BY PERCEIVED APPROPRIATENESS

Given the difference in perception due to application appropriateness, we are interested to see if participants' opinion of how appropriately the application behaved, and how much they agree with its inference were affected by intelligibility. These self-reported, repeated measures for every scenario were obtained with the following questions:

Perceived Appropriateness with "How appropriately or inappropriately did the application behave in this situation?" (7-point Likert scale)
Agreement with "How much do you agree or disagree with the application's inference, given how easy or difficult it is to infer this?" (7-point Likert scale)

We did not compare perceived certainty, because, as expected, it varied independently of appropriateness.

We fit a mixed model with: agreement as dependent variable, appropriateness and agreement as independent variables, appropriateness × agreement as an interaction effect, and participant as a random variable (nested in appropriateness and agreement). Our results (see Figure 8.7 and Table 8.6) show that participants tended to agree with the application when they perceived it behaved appropriately, and *vice versa*. When participants felt the application behaved inappropriately (<0), those with intelligibility agreed with the application more than those with None ($A_{\neg a}$). However, when participants perceived the application behaved very appropriately (2-3), participants using an application with low certainty and intelligibility agreed less with it than those with None (Figure 8.7, Right; A_a).



All Actual Certainties (50-100%)

Low Actual Certainty (50-70%)

F(12,4984)=2.92, p<.001

F(12, 1645)=3.31, p<.001

Figure 8.7. Agreement across Perceived Appropriateness, grouped by actual certainty. The effect of finding 4h is only significant for low actual certainty.

Actual Certainty	**All (50-100%)**		**Low (50-70%)**	
Appropriateness	**Not (<0)**	**High (2-3)**	**Not (<0)**	**High (2-3)**
None *vs.* Full	p<.01	p<.05	p<.01	p<.01
None *vs.* Certainty	p=.052	p=n.s.	p=n.s.	p<.05

Table 8.6. Contrast between Intelligibility types for low and high appropriateness grouped by actual certainty.

8.9.5 SUMMARY OF FINDINGS

We summarize our findings in terms of our hypotheses. Participants with Full intelligibility gave more correct reasons of how the application works, than those without (satisfies **H2**). *Finding* C_h satisfies **H1a** that intelligibility improves user impression of a context-aware application if its certainty is high. This is more pronounced when the application behaved inappropriately ($C_{\neg a,h}$) than appropriately ($C_{a,h}$). Conversely, *finding* C_l satisfies **H1b** that intelligibility harms user impression if its certainty is low; particularly, when the application behaved appropriately ($C_{a,l}$, A_a). However, participants with intelligibility disagreed less with the application inference when they felt that it behaved inappropriately ($A_{\neg a}$). To gain better insights into our results, we ran a follow-up study where we engaged participants face-to-face.

8.10 FOLLOW-UP: THINK-ALOUD STUDY

At this point, our results positively support our hypotheses that intelligibility *exaggerates* the perception of certainty compared to not receiving any explanation. However, this could be because our participants with intelligibility could just be copying the certainty value they were shown (see Table 8.4). Do participants mindlessly copy these values, or do they weigh their opinion with previous experiences with the application (from previous scenarios)? Furthermore, *finding* A_I suggests that Full intelligibility provides some additional benefit of improving the perception of certainty than showing Certainty-only. How does Full intelligibility help to reinforce the Certainty information provided?

8.10.1 METHOD AND PROCEDURE

Answering these questions will help us examine the link between the information provided through Certainty-only and Full intelligibility, the user's understanding, and their subsequent impression of the application. It will also help us explore whether and how Full intelligibility influences the user compared to Certainty-only. To explore this, we ran a follow-up *think-aloud* study where we presented all three intelligibility conditions *within-subject*, focusing on a subset of Certainty conditions (low: 50%; high 90%). We continued to use both applications (*between-subject*) due to their differences in complexity, and participant reliance on their explanations. Hence, we have four conditions. Due to the time-consuming nature of the think-aloud study (one-hour long), we presented only two scenarios (S6, S9) to our participants, counter-balanced for application correctness. S6 has been described in Table 8.1. For S9, LocateMe correctly infers the user in Meeting Room B and automatically loads the meeting agenda; HearMe correctly infers conversation during a group meeting, and allows the user to retrieve the audio and save it. In the follow-up study, the application always behaves incorrectly for S6, but correctly for S9, regardless of certainty.

We recruited two participants per condition (total 8), 4 females, mean age 28.1 years old (21 to 58). We presented three iterations of the survey starting with None, Certainty, then Full, so as to avoid a training effect. Each scenario has the same format and questions as the original survey. Additionally, we asked them to think-aloud and provide reasons for their answers. This way, we learned about how they thought the application made its inferences, and how they constructed opinions of the application's behavior. We used paper surveys, so participants could refer to previous surveys, compare previous phone displays, their previous answers, and discuss why they changed or did not change their opinions. Table 8.7 shows which conditions participants P1 to P8 were in. We discuss our findings in the next section in the context of our original quantitative results.

Certainty	**Low** (S6:52%, S9:49%)				**High** (S6:89%, S9:92%)			
Application	**LocateMe**		**HearMe**		**LocateMe**		**HearMe**	
Participant	1	5	2	8	3	5	4	7

Table 8.7. Distribution of participants in think-aloud study. Each participant saw S6 (appropriate behavior) and S9 (inappropriate), iterated within-subjects with intelligibility types in the order: None, Certainty, Full.

8.11 DISCUSSION

We discuss the results from both experiments in terms of how intelligibility affects understanding (H2), and how it affects users' impression of context-aware applications (H1).

8.11.1 H2: INTELLIGIBILITY INCREASES UNDERSTANDING OF CONTEXT-AWARE APPLICATIONS

As expected, Full intelligibility allowed participants to better express an understanding of the applications. This was particularly significant for HearMe, because the explanations listed relevant factors, increasing the participants' vocabulary to describe how the application works. In the think-aloud study, participants could analyze and interpret the values of HearMe's sensed factors, and their corresponding weights of evidence, and LocateMe's bubble visualization. However, because the input factors were not explicitly stated in LocateMe as they were for HearMe, participants gave reasons for how LocateMe works by describing names of technologies, *e.g.*, GPS, "position-specific sensing" (P3), a "grid in the building" (P6), or in terms of the situation, *e.g.*, signal blocked by nearby stairs (P5), or improved signal because of proximity to windows (P5). Full intelligibility only marginally increased the correct ideas that participants had about how LocateMe works. For HearMe, without Full intelligibility, participants considered the "noisiness" of the audio, along with speaker identification (especially that of the user) as the most important factors for inference, but with Full intelligibility, they tended to discard their original understanding and described the inference in terms of the factors shown. We can interpret the differences between the applications as due to their *complexity* and the users' *familiarity* with them. These findings reinforce those in Section 4.7.1 [Lim, Dey, and Avrahami, 2009] that prior knowledge about an application domain (in this case, LBS) reduces the impact of intelligibility on understanding.

By providing more information, intelligibility also helps provide participants with an increased awareness of what the application was inferring, and what it understood. This consequently impacted their impression of it.

8.11.2 H1:IMPACT OF INTELLIGIBILITY ON USER IMPRESSIONS

Without Intelligibility, MTurk participants are influenced by whether the application behaved appropriately to perceive its certainty (see Figure 8.5, Right). They perceived a modestly high certainty (~85%) when it is behaved appropriately, and how often it behaved appropriately (Figure 8.4), but perceived a low certainty otherwise (~60%). Their overall perceived certainty is also impacted by the cumulative application behavior, gently increasing from ~70 to ~90% as actual certainty increases from 50 to 100% (Figure 8.5). With intelligibility, participants' perceived certainty aligned more closely with the actual certainty.

However, do participants just copy the application certainty (as suggested in Table 8.4)? In the think-aloud study, though influenced by the displayed value, *all* participants did not outright adhere to it. They continued to be influenced by their perception of how difficult it was to make the inference, and whether the application behaved appropriately, but adjusted their certainty rating depending on the presented value. Hence, if a low certainty was presented, participants lowered their certainty estimate, and while if a high certainty was displayed, participants raised their certainty estimate, but not all the way to the presented value for both cases. Furthermore, participants reevaluated their certainty rating when given Full intelligibility. Next, we discuss and interpret our results (shown in Figure 8.6 and Figure 8.7) in terms of hypotheses H1a and H1b. We found a caveat to H1b which we denote as H1b'. Table 8.8 summarizes these positive and negative impacts that intelligibility has on user impressions.

		Certainty	
		Low	**High**
	Overall	**Harmful (H1b)** Decreases perceived overall accuracy (*finding C_l*).	**Helpful (H1a)** Increases perceived overall accuracy (*finding C_h*).
Behavior Appropriateness	**Appropriate**	**Harmful (H1b)** Decreases perceived accuracy (*finding $C_{a,l}$*), and Decreases agreement with inference (*finding A_a*).	**Helpful (H1a)** Increases perceived accuracy (*finding $C_{a,h}$*).
	Not	**Helpful (H1b')** Increases agreement with inference (*finding $A_{\neg a}$*).	**Helpful (H1a)** Increases perceived accuracy (*finding $C_{\neg a,h}$*).

Table 8.8. Impact of Intelligibility on user impressions of a context-aware application depends on application certainty and whether it behaved appropriately.

8.11.2.1 H1A: INTELLIGIBILITY INCREASES USER IMPRESSIONS OF CONTEXT-AWARE APPLICATIONS WITH HIGH CERTAINTY

For context-aware applications with **high certainty**, our results verify previous findings of Chapters 4 [Lim, Dey, Avrahami, 2009] and 5 [Lim and Dey, 2009] that intelligibility improves users' impression of the applications (*finding C_h*). With intelligibility, participants perceived a higher certainty from the application, particularly when it **behaved appropriately** (*finding $C_{a,h}$*). After seeing the Certainty-only intelligibility, P4 raised her original rating (85% with None) to "just below" what was shown (92%), despite feeling that HearMe was “overconfident,” because of her “overall understanding” of the conversation that she heard in the audio clip. With Full intelligibility, participants felt the explanations “reinforced” the high certainty (P3), or even raised their certainty of the application (P6).

Intelligibility had a more significant impact on perceived certainty if the application **behaved inappropriately**, since MTurk participants had a lower baseline certainty rating (about 60% instead of ~85%). Though not to as high a level for appropriate application behavior, intelligibility raised their confidence rating by a larger margin (by 15% to ~75%; *finding $C_{\neg a,h}$*). With Certainty-only intelligibility, P3 liked that LocateMe was "honest," and trusted it more. P4 felt that HearMe "would know its own certainty better than [she] would" and raised her certainty to 75-85%, which was between what she had imagined and what was presented. P6 insisted that LocateMe's certainty should not have been so high, but raised her rating by 5%. With Full intelligibility, participants reevaluated their opinion. P4 and P7 were more accepting of HearMe's certainty, and raised their ratings.

8.11.2.2 H1b: Intelligibility Decreases Impressions of Applications with Low Certainty when they behave Appropriately

For context-aware applications with **low certainty**, intelligibility revealed how uncertain they were, and compromised the impressions participants had of them (*finding* C_l). This was particularly notable when the application **behaved appropriately** (*finding* $C_{a,l}$). In the think-aloud study, participants were surprised to discover the low certainty. For LocateMe, P5 thought that the location in S9 was easier to infer than in S6, and felt that the certainty should have been higher (60%) than the presented 49%. For HearMe, P2 felt that the conversation in S9 was "so clear" that the certainty should be higher at 60-65%. Furthermore, the unexpectedly low presented certainty caused participants to disagree more with the application inference (*finding* A_h). P1 and P5 lowered their agreement rating from 7 (None) to 2 (Certainty), and P8 from 6 to 4. With Full intelligibility, P8 became convinced by HearMe's displayed 50% certainty by examining the bar chart of weights of evidence and noting they were very balanced; she consequently lowered her certainty rating.

8.11.2.3 H1b': Intelligibility Increases Impressions of Applications with Low Certainty when they behave Inappropriately

Contrary to H1b, intelligibility was helpful for an application with **low certainty** when it **behaved inappropriately** (*finding* A_a), even though it did not influence perceived certainty (*finding* $C_{\neg a,l}$). Participants appreciated the difficulty of inference, forgave the application, and disagreed less with it (A_l). P5 conceded that it was "very difficult" for LocateMe to estimate the certainty, and thought "it got it almost right"; she agreed more with the application, changing her rating from 4 (None) to 5 (Certainty). With Full intelligibility, participants more clearly saw how uncertain the applications were: large margins of error (LocateMe), or a high amount of ambiguity (HearMe). This allowed P8 to understand how HearMe "misjudged the environment," and "agree with its logic based on the parameters."

8.12 DESIGN RECOMMENDATIONS

		Certainty	
		Low	High
	Overall	**None** Prioritize improving accuracy instead of providing intelligibility.	**Intelligible** Provide Intelligibility.
Behavior Appropriateness	Appropriate	**Automatically Hide** User is unlikely to ask questions if behavior is as expected.	**Automatically Show** Provide intelligibility automatically if certainty is high.
	Not	**On Demand** User will likely ask questions and receive helpful explanations.	**+ On Demand** Users may ask more questions to receive more helpful explanations.

Table 8.9. Summary design recommendations of when and how to provide intelligibility given the uncertainty in a context-aware application.

There are two ways to apply our findings in terms of application certainty: regarding *overall* certainty, or *per situation* certainty. Considering overall certainty, our findings recommend providing intelligibility as long as the application usually has high certainty. However, our findings caution that intelligibility is harmful if certainty is too low, so intelligibility should not be provided for such applications; their certainty should be improved first. While the precise threshold for what is a *sufficiently* high overall certainty depends on the application and domain, our results (see Figure 8.4) suggest it falls within the range of about 80-90% for a non-critical, "everyday" application. Table 8.9 summarizes our design recommendations from this study.

Considering certainty per situation, we note that an application that usually has high certainty may still occasionally have situations with low certainty. Fortunately, in the majority of times with high certainty, we still recommend providing intelligibility even if it is ultimately wrong and behaves inappropriately. On the other hand, our recommendation is not immediately clear with low certainty. The impact of intelligibility on impression also depends on whether the application behaves appropriately.

If the application behaves appropriately, showing intelligibility compromises the original good impression the user may have had, causing her to lower her impression. If it behaves inappropriately, intelligibility can help her realize how difficult the inference task is, and improve her impression. Unfortunately, an application will not be able to know if it will act appropriately beforehand. It will be safer to not show intelligibility before it acts, when certainty is low. After it acts, if the user asks questions, especially if a *why not* question, it is likely the application behaved inappropriately, where it is beneficial to show intelligibility. Therefore, just show intelligibility *on demand*, when situation certainty is low. Our results (see Figure 8.6, Left) indicate that our participants have a baseline belief that the application is about 80-85% certainty when it behaved appropriately. This suggests that, for each situation, a context-aware application may safely provide intelligibility automatically when it is at least 80% certain, but should provide intelligibility on demand when it is less certain.

Carefully designing explanations can provide an alternative solution to deal with intelligibility as a double-edged sword for low certainty. Intelligibility should focus on convincing the user how difficult the inference task is, and how the application is intelligently tackling it, rather than implying that the application is incompetent. This could mean not revealing the low certainty in explanations. For example, HearMe's Sensed Factors visualization explains what *input values* it knows, but does not betray HearMe's uncertainty.

8.13 CONCLUSION AND FUTURE WORK

We have described a large controlled study investigating the impact of intelligibility on understanding and user impressions of context-aware applications with varying certainty from low to high. This was conducted using lab-based, scenario-driven surveys of two context-aware applications (location-aware, and sound-aware). Our results show that intelligibility can positive or negatively impact user impressions, depending on the application's certainty and behavior appropriateness. Intelligibility is helpful for applications with high certainty, but it is harmful for applications with low certainty, because the user loses even more trust in its capability. Still, intelligibility can help users appreciate and forgive applications if they behave inappropriately and have low certainty. This work explicitly cautions the necessity for a context-aware application to be sufficiently certain before it leverages intelligibility.

In this study, we have focused on a *passive* display for intelligibility, and did not have users act on the information; they could only use intelligibility to judge their impression of the application. However, users could also *interactively* use intelligibility for debugging and finding out why the application faltered (*e.g.* [Kulesza *et al*., 2009]). Perhaps, this could make intelligibility useful instead of harmful to user impression.

This work provides a stepping stone to understanding how intelligibility affects a user's impression of a context-aware application. For future work, we plan to gain more lucid and nuanced insights into the use of intelligibility and how that affects users in real-world situations through deploying an intelligible prototype in a longitudinal field trial. In preparation for a field deployment, we evaluated a second version of Laκsa, an intelligible context-aware mobile application, for its usage and usefulness for improving user understanding. We describe this study in Chapter 9.

9 EVALUATING THE USAGE AND USEFULNESS OF INTELLIGIBILITY

ABSTRACT. Intelligibility has been proposed to help end-users understand context-aware applications with their complex inference and implicit sensing. Usable explanations can be generated and designed to improve user understanding. However, will users be willing to use these intelligibility features? How much intelligibility will they use, and will this be sufficient to effectively improve their understanding? We present a quasi-field experiment of how participants used the intelligibility features of a fully-functional intelligible context-aware application. We investigated how many explanations they willingly viewed, how that affected their understanding of the application's behavior, and suggestions they had for improving its behavior. We discuss what constitutes successful intelligibility usage, and provide recommendations for designing intelligibility to promote its effective use.

9.1 INTRODUCTION

Much of our work on investigating the impact of intelligibility had focused on questionnaire studies and ‘paper’ prototypes of realistic albeit fictitious context-aware applications (Chapters 4, 5, and 8). With the Laκsa prototype (Chapter 7), we sought to increase realism in investigating intelligibility with an interactive prototype. While that work provides a crucial step for designing intelligibility to be more usable and interpretable, it stopped short of *evaluating* the impact of intelligibility on users. In Chapter 8 [Lim and Dey, 2011b], we investigated the impact of intelligibility on understanding and impression, but this was studied with questionnaires and ‘paper’ prototypes rather than an interactive prototype. Furthermore, intelligibility was shown “always on” to participants, so they were biased to look at the explanations. This leaves open the research questions: even if intelligibility can improve user understanding and trust, will users want to use it, and, if so, how much? Moreover, given how much they do use, how much will that improve their understanding of context-aware applications?

Related work has explored the impact of explanations on end-users as they used context-aware systems. Rukzio *et al.* [2006] evaluated a mobile phone automatic form filler in a lab study, and found that *"visualizing the uncertainty of the system was mostly not used nor was it helpful."* Tullio *et al.* [2007] evaluated an intelligible interruption door display over six weeks, and found that users were able to *"attribute concepts of machine learning to their system,"* but had difficulty remembering relevant features. Cheverst *et al*. [2005] deployed the Intelligent Office System that provided explanation visualizations of rules and confidence. However, regarding explanations, their evaluation focused on eliciting user preference about visualization format, not on their impact. Welbourne *et al.* [2010] investigated the use of Panoramic that is able to explain location with timeline visualizations. However, their evaluation involved participants investigating realistic, but fictitious, data. Vermeulen *et al*. [2010] conducted a pilot user study of PervasiveCrystal in a simulated museum with five participants, who *"were able to use the questions interface to find the cause of events"* of three tasks. Kulesza *et al*. [2009] evaluated the usage, debugging, and understanding of the Why and Why Not explanations of email filtering application.

This chapter adds to this body of work evaluating intelligible context-aware systems by explicitly measuring the *usage* of intelligibility in a high-fidelity prototype that provides over nine explanation types (*e.g.*, Certainty, Why, Why Not, What If) for three context types (Availability, Place, Sound). We iterate on Laκsa (see Section 7.2) to investigate usage under realistic situations with real-time application behavior and automatically generated explanations. We also investigate the impact of this usage on user understanding of the application's inference. Our contributions provide insight into the usefulness of intelligibility by investigating:

1. How much participants *use* intelligibility in a real context-aware application,
2. Their *opinion* of the *usefulness* of the explanations to understand application behavior and situations, and
3. How *useful* their use of intelligibility is on understanding and handling of these situations.

The rest of the chapter is organized as follows: we articulate our objective to explore the usage of intelligibility, and our hypothesis that increased intelligibility usage will improve user understanding. We developed a functional intelligible context-aware prototype for this study, which we describe next. Following that, we elaborate on the quasi-field experiment we conducted, where participants engaged in "everyday" scenarios *in-situ* using the prototype. We follow this with the results showing how participants used intelligibility in our prototype and how that improved their understanding of application inference for each scenario. Finally, we discuss design implications due usage patterns and constraints, and how to encourage users to use more intelligibility to further improve their understanding.

9.2 OBJECTIVES AND APPROACH

We have two objectives for this study: one explorative and another hypothesis-driven.

1) Exploring the usage of Intelligibility. We aimed to investigate *how* users use intelligibility when facing different scenarios, *how much* they use, and for *how long*.

2) Hypothesis: Increased usage of Intelligibility will improve user understanding. We hypothesize that using intelligibility more will help users better understand application inferences and the current situation.

To accomplish these objectives, we conducted a quasi-field study where participants used a fully interactive, intelligible context-aware application under real-world, "everyday" situations (similar to [Roto *et al*., 2004]). To improve ecological validity of our results, we minimized interference from the experimenter by logging and analyzing UI events [Hilbert and Redmiles, 2000] of intelligibility usage (without thinking aloud), and post-incident interviews. This experimental set-up strikes a balance between controlling for critical incidences, and allowing participants to use intelligibility naturalistically. Note that in this work, we do not claim to cover a comprehensive set of situations or motivations under which intelligibility may or may not be used significantly. However, we seek to gain an initial insight into how intelligibility may be used in a context-aware application reacting to a real physical environment.

We next describe the intelligible context-aware prototype we developed and employed to study intelligibility usage.

9.3 LAKSA2 PROTOTYPE

Mobile phones allow people to keep in touch with others and be easily reachable. However, there are times when receiving calls are inappropriate, as they are socially disruptive (*e.g.*, in meetings and movie theatres), or they interrupt productive work. Users can manually silence their phones, but they may forget to reset their phones to ring again afterwards [Milewski and Smith, 2000]. Hence, it will be useful if the phone can automatically set the ringer mode (*e.g.*, [Kern and Schiele, 2006; Rosenthal, Dey, and Veloso, 2011]). Also targeting this compelling application problem domain, we have developed Laκsa2, a mobile application that senses various contexts (Place, Sound, and Schedule) about the user to automatically infer her Availability, and set her phone's ringer mode. Our focus is the use of Laκsa2 as a platform to explore the use of intelligibility in a context-aware application. Next we describe Laκsa2's contexts.

Availability: *Available, Semi-Available, Unavailable* — is inferred from rules regarding the following three factors.

Place: *Office, Café, Library, etc.* — represents the semantic location of the user. It is inferred by sensing latitude and longitude from the Android Location API (uses GPS, Wi-Fi, and cell tower positioning), and matching to pre-specified named places. The user's sensed location is modeled as a radial Gaussian, with decreasing likelihood further away from the latitude and longitude coordinates (similar to LocateMe in Section 8.5.1). Laκsa2 stores a list of named places with coordinates and size (circle radius) to compare against to infer whether the user could be at each place. Each Place is inferred with different certainty based on how much the user's estimated location area "overlaps" with the area of the named place: more overlap leads to higher certainty.

Sound: *Talking, Music, and Ambient Noise* — represents the sound activity that Laκsa2 recognizes from what the phone's microphone hears. Inferences come from a naïve Bayes classifier trained on sound samples. Features extracted are similar to SoundSense [Lu *et al*., 2009]: *e.g.*, mean of power, low-energy frame rate, spectral flux, and bandwidth. These are renamed to lay terms that end-users can understand.

Schedule: *Personal, Work, Unscheduled* or *Other Event* — represents the user's Google Calendar, and, in particular, which calendar the current event is in.

9.3.1 INTELLIGIBILITY FEATURES

Having defined the context types, we make Laκsa2 intelligible so that users can understand what it knows and how it makes inferences. Laκsa2 provides explanations from the Intelligibility Toolkit (Chapter 6) ported to Android:

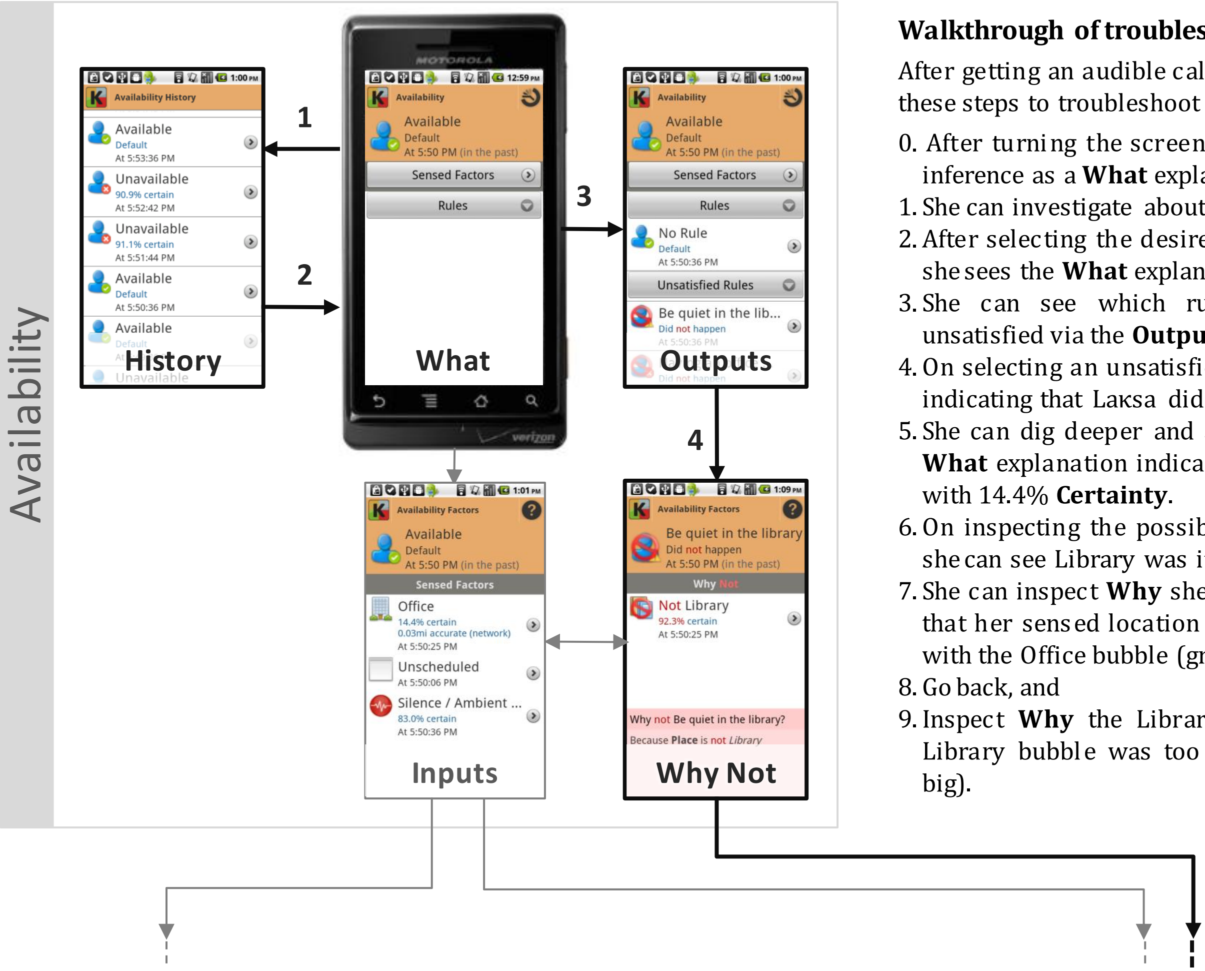
1. **What** is the inference for the context? With how much **Certainty**? **When** was this value inferred?
2. **History**: what was the inference at time *H*?
3. **Inputs**: what details affect this context? (Factors, input features, related details, *etc.*)
4. **Outputs**: what values can this context be inferred as? With how much **Certainties** are these values inferred?
5. **Why** was this value inferred?
6. **Why Not (Why Alt)**: why wasn't this inferred primarily as *Y*, instead? (Note that alternative values may have been inferred, but not necessarily as the first choice.)
7. **What if** the factors are different, what would this inference be? (Requires user manipulation)
8. **Description**: meaning of the context terms and values.
9. **Situation** of what was happening to affect the inference to provide a ground truth of what was being inferred (*e.g.*, playing an audio clip of what was heard).

Some explanation types have been aggregated to reduce the number of questions users need to ask (*e.g.*, What + Certainty + When, Outputs + Certainties). For simplicity, What If was only provided for Availability. Also, Schedule does not have particularly expressive explanations, because it is easy to understand calendars and events.

9.3.2 Design Iterations and Updates

While this iteration bears many similarities to the original Laκsa prototype (Section 7.2), its design and functionality has been significantly refined and it uses streamlined questioning to be simpler for users. Further feedback from colleagues, who are HCI researchers, helped make the user interaction more consistent throughout the application, and reduced the application functionality so that users can grasp its concepts within a two-hour study. Therefore, we removed the Motion context of Laκsa v1. Laκsa2 also supports more explanation types that are relevant to realistic use, such as History and Situation. Finally, Laκsa2 is not a social-awareness application like its predecessor, and it was fully deployed on a mobile phone instead of a Tablet PC.

For the rest of this chapter, we will refer to Laκsa2 as Laκsa, unless otherwise stated.



Walkthrough of troubleshooting S3

After getting an audible call in the Library, a participant may follow these steps to troubleshoot why Laκsa did not silence the phone.

0. After turning the screen on, she will see the current Availability inference as a **What** explanation.
1. She can investigate about a specific past event with **History**.
2. After selecting the desired event, in this case, one at 5:50:36 PM, she sees the **What** explanation of her Availability at that time.
3. She can see which rules were triggered and which were unsatisfied via the **Outputs** explanation.
4. On selecting an unsatisfied rule, she sees a **Why Not** explanation indicating that Laκsa did not think she was at the Library.
5. She can dig deeper and ask about the Place inference to see the **What** explanation indicating Laκsa thought she was at the Office with 14.4% **Certainty**.
6. On inspecting the possible places with the **Outputs** explanation, she can see Library was its second choice.
7. She can inspect **Why** she was inferred to be at the Office, and see that her sensed location (concentric blue bubbles) overlaps a lot with the Office bubble (green);
8. Go back, and
9. Inspect **Why** the Library had a lower **Certainty** (because the Library bubble was too small, while the Office bubble was too big).

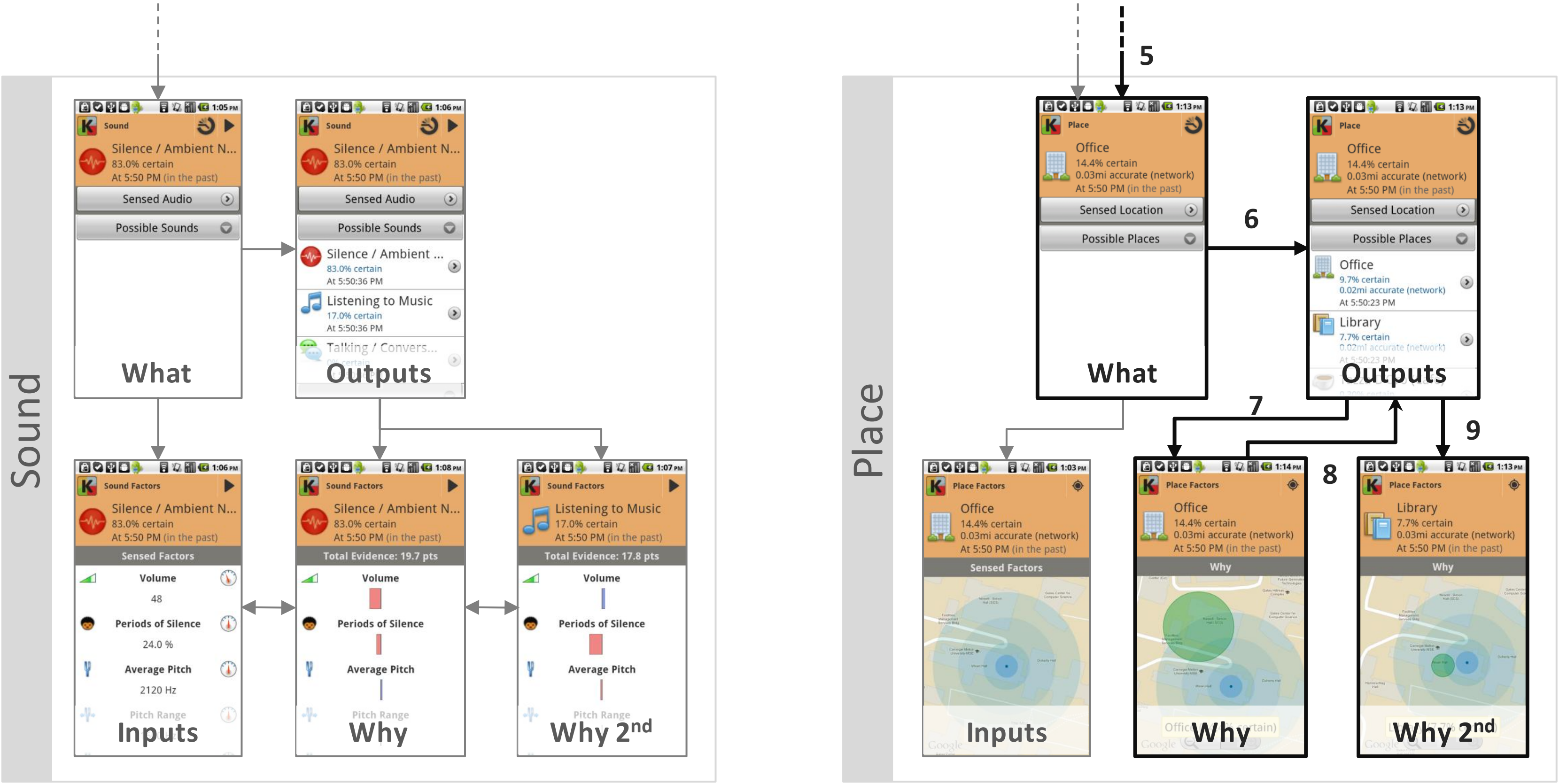


Figure 9.1. Screenshots of Laκsa showing several explanation types of the upper-tier context Availability, and lower-tier contexts Sound and Place. Arrows show how a user can transition from one explanation to another. The bold trace indicates how a participant may explore the intelligibility features in nine steps to troubleshoot Scenario 3 about the phone ringing in the Library. See Section 7.3 for a description of some of the UI design.

9.3.3 IMPLEMENTATION AND USER INTERFACE

We developed Laκsa2 for Android 2.2 Froyo (API level 8), and deployed it on the Motorola Droid for the user study. Sensing for location, calendar events, and microphone audio were performed using background services on the phone every 30 seconds. Higher-level inferences for Place, Schedule, Sound, and Availability are computed in the background, in response to each sensed instance. To recognize sounds, we used a port of Weka for Android[3]. We also partially ported the Intelligibility Toolkit [Lim and Dey, 2010] to Android, to support the *querying* for various questions, *generation* and *reduction* of explanations about the contexts, and *presentation* of the explanations in various graphical and textual formats. Unlike Laκsa v1 (see Section 7.4), the Laκsa2 prototype is fully implemented on the mobile phone for sensing, inferring, reacting, and displaying explanations. Figure 9.1 shows several screenshots of the Laκsa2 prototype with a walkthrough example of how to use it. Appendix H shows more screenshots of the application and several explanation types. Each explanation is viewed as a *page view*. Users can transition from one to another by clicking on buttons, menu items (from the options menu), and flinging (swiping). Some explanations allow scrolling to see more details.

9.4 SCENARIO-DRIVEN QUASI-FIELD STUDY

To explore the use of the intelligibility features in Laκsa, we conducted a controlled scenario-driven user study, where participants encountered situations that may arise with the use of Laκsa. We were interested in whether and how participants used the intelligibility features to understand the application behavior. We conducted a quasi-field study rather than a field deployment to (i) present participants with controlled critical incidences, and (ii) observe and measure their subsequent behaviors due to these incidences. It otherwise would have been difficult to know when critical incidences occurred in the field, or why.

9.4.1 PROCEDURE

An experimenter first briefed the participant about the study, and presented her with printed instructions. These describe how to use the Android phone, Skype (for receiving or checking calls), and Laκsa's functionality and interface. The experimenter gave a walkthrough of Laκsa,

[3] Weka for Android. https://github.com/rjmarsan/Weka-for-Android. Retrieved 26th August 2011.

demonstrating its features and how to interpret them. We provided participants with availability rules that Laκsa was pre-programmed with: four rules setting availability to Unavailable and Semi, and any other case as Available (*e.g.*, if the user is in her office and Laκsa hears talking, her status is set to Unavailable). Participants did not touch the phone until the first scenario (S1).

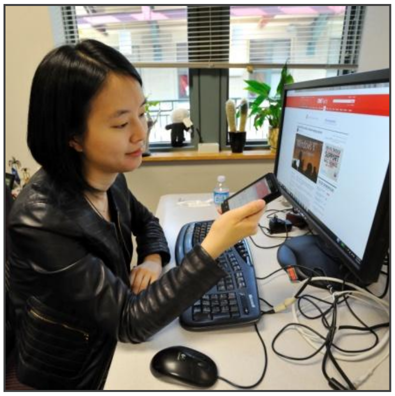
The participant was instructed that she works with equally ranked coworkers in several offices, and that they work together on a team project. She was provided with the following motivation: she needs to evaluate Laκsa as a newly acquired application, which can improve her team's productivity by moderating interruptions. She is tasked with the overall goal of determining when Laκsa behaved appropriately or not, and figuring out how to improve its future behavior by (i) editing availability rules, (ii) changing lower-level settings (*e.g.*, size of Place bubbles), or (iii) changing behavior (*e.g.*, lower the music volume). She would also be responsible for subsequently teaching her coworkers how to best configure Laκsa. After reading the instructions, the participant begins the scenarios.

9.4.2 Controlled In-Situ Scenarios

The user study was scenario-driven to expose participants to situations they may encounter with Laκsa. To increase the visceral quality of the scenarios, each scenario is set up by bringing the participant to the necessary places (Office, Library, or Café) and asking her to carry out an initial task, *e.g.*, looking for a library book (S3). The experimenter shadowed the participant for every scenario. The participant engaged in the activity for a few minutes to become absorbed in the situation. Critical incidences were triggered by the experimenter calling the participant's phone (if necessary), and presenting her with a printed flash card describing what was happening, and any associated dialog with coworkers during the phone call. The participant was free to interact with Laκsa as much or as little as she wished, and prompted to *not* think aloud. For each scenario, after she was done looking at Laκsa, she turned off the screen, and the experimenter conducted a structured interview with audio recording. She was asked about her opinion of the situation and the application, her understanding of how Laκsa was making inferences, and any suggestion she may have for improving its behavior.

We employed four scenarios to span three situational dimensions: (i) Exploration / Verification (S1) of Laκsa's functionality and explanations; (ii) Fault Finding (S2, S3), where Laκsa behaved inappropriately, and participants had to troubleshoot it; and (iii) Preemptive Exploration (S4) where participants investigated a potential future situation.

S1: Talking in the office. Training session where the participant learned Laκsa's core features and explanations. She could explore Laκsa as much as possible to familiarize herself with the application UI and explanations.



S2: Missed call while reading news and listening to music. The participant is asked to read any news articles they fancy from www.cnn.com while they listen to a song[4] through the computer speakers. Meanwhile, she received multiple calls from a coworker, but she misses the first few calls. The experimenter actually called the participant's phone but it did not ring the first few tries since Laκsa automatically silenced its ringer. Near or at the end of the song, the phone would ring again and the participant would notice the call. Through a flash card, the participant learned that her coworker, Damien, was frustrated from trying to call her repeatedly over the past three minutes; he would like her to check her email, and fix her phone. The email pertained to finding a library book to review for their shared project. Laκsa had mis-inferred Sound as Talking instead of Music, and behaved inappropriately by silencing the phone.

S3: Phone interruption in the library. As a follow-up to Damien's email, the participant walked to a nearby library to search for the book, and read it. Meanwhile, the experimenter called her phone again, causing it to ring audibly in the quiet library. This simulated a coworker calling. Laκsa had mis-inferred the participant's Place as still in the Office instead of Library, and behaved inappropriately by allowing the phone to ring.

[4] Sound of Silence by Simon and Garfunkel. http://youtu.be/eZGWQauQOAQ Retrieved 17 September 2011.

S4: Preemptively checking availability in café. The participant received a flash card describing that she frequents a café (in a nearby building), and should check whether she will be able to receive calls there. Participants could sit where they were and check for explanations in such a hypothetical situation or visit the care, but they were not prompted what to do to achieve this objective.

9.5 Measures and Data Preparation

We are interested in measuring how useful intelligibility was for the participants in terms of how much they used, and how that impacted their understanding of Laκsa, and their suggestions on how to control it to resolve any issues.

9.5.1 Smartphone Ownership

To control for technical experience among participants, we asked participants whether they owned smart phones (**Smartphone Ownership**) and for how long. This measure may distinguish users who are more technology savvy and more interested to explore new technologies, from those who are not.

9.5.2 Usage of Intelligibility

To measure intelligibility usage, we logged when participants viewed each explanation page in the UI. This allowed us to measure, for each scenario, which explanation types each participant viewed, how many (**# Explanation Types**), when they were viewed, for how long (**Duration**), how often (**View Count**), and their sequence order (**Step** number). We built a network graph for each participant scenario to illustrate the *sequence diagram* of how he used intelligibility (*e.g.*, Figure 9.1). With these we can observe general patterns of use, and identify errors in the logging. Since participants may view certain pages only to get to another page (*e.g.*, transitioning through Availability Inputs to get to explanations about Place), they may only view them for a very brief duration. Hence, we filtered out views with durations < 1 second. Additionally, intelligibility usage patterns may also affect what a user learns of the application. One such metric is the **Context Ratio** of how many explanation types of deeper contexts (Place and Sound) are viewed compared to that of the shallower context (Availability). Having generated these metrics, we wanted to investigate if they influence user understanding.

9.5.3 USER UNDERSTANDING AND SUGGESTIONS FOR CONTROL

We measure how well participants understood the application behavior and scenario circumstance by transcribing audio from interviews along with notes. To measure their understanding, we asked them what they understood about what Laκsa knew and how it was reasoning. The transcript was coded into units of *beliefs* to characterize their mental models, using the coding scheme in Table 9.1. Their statements are a lower bound of their understanding, since they may not have said everything they believed. To derive a single metric of understanding, an **Understanding Score** is calculated for each participant scenario by adding all 7 codes for both Place and Sound (Max=14). This score represents the breadth and depth of understanding a user has for the scenario.

Another measure of how well participants understood Laκsa is how many effective **Control Suggestions** they provided to overcome any issues or problems in the scenarios. A weighted **Control Score** is calculated from summing scores for each code in the scheme in Table 9.2. This score represents the number and effectiveness of suggestions provided for the scenario. Partially effective suggestions are given only half a score. These suggestions may have side-effects that compromise application performance in other situations (*e.g.*, adjusting the weights of a sound factor to influence recognition).

Note that so as not to prime or mislead participants with knowledge about how Laκsa makes inferences, we do not ask multiple choice comprehension questions. Instead, we record open-ended descriptions from participants, which we code to determine their level of understanding.

Code		**κ**		**Description / Example Transcripts**
		Place	**Sound**	
U_1	Value	.83	.89	Indicated knowledge of the inferred value of the factor.
U_2	Alternative Values	.78	.85	Indicated knowledge of other inferred (2nd, 3rd, *etc.*) or uninferred values. Compared different values that were inferred differently, e.g., P01S2 *"Talking (evidence=85.4) very close to music (84.?). Could have gone any way."*
U_3	Certainty	.94	.87	Described certainty of inferred value, *e.g.*, P02S3: *"It was 9.3% certain I was at the Office"*; P03S3: *"blue bubbles were too big."*
U_4	Inputs	.85	.94	Mentioned at least one input feature / factor of the context, e.g., Pitch, Periods of Silence, *"the blue bubble was directly over the Library building."*
U_5	Model	.86	1	Described the mechanism for inferring the factor, *e.g.*, P17S3: *"It looked like it was actually probably closer to the library, but since the library bubble was very small then it calculated the probability was very low."*
U_6	Technical	.90	0*	Provided a deep technical mechanism for the inference not explicitly described in the explanations, *e.g.*, P18S3: *"It seems to be based on its Wi-Fi connection, and ... because it said networking and it gave the location badly and we're deep inside a bunch of concrete and metal, so the GPS shouldn't be working right now."*
U_7	Situation Justification	.80	.94	Provided a situational justification for the phone's inference that was not from the intelligibility UI, *e.g.*, P02S2: *"The music was much mellower, and they were really singing"*; P07S3: *"We were very close to previous location [Office], not easy to pinpoint current place [Library]."*

Table 9.1: Coding scheme for user understanding. Participants' mental models were decomposed into beliefs based on what they explicitly said and tacitly implied. Each scenario may have multiple codes, each either 0 or 1 indicating whether the correct corresponding beliefs were expressed. We only coded for Place and Sound factors, since participants' understanding of Availability can be derived from their understanding of these. Inter-coder reliabilities (κ) for each code were calculated with a 35% random sample of the scenarios by a second coder. * denotes apparent low reliability due to low count.

Code		κ	Description / Example Transcripts
C_1	Availability Rules	.89	Proposed a new rule, editing an existing rule, or deleting one, *e.g.*, delete rule "Someone's Talking"; add rule "Office + Music → Vibrate"
C_2	Place Settings	.90	Suggested to adjust the bubbles of Places by enlarging, shrinking, or moving them. Suggested to threshold blue bubbles to calculate overlap between sensed location and Places.
C_3	Sound Settings	.89	Suggested to adjust a feature weight to tweak inference; Suggested to expand training data, *e.g.*, P10S2: "Teach it more about music by inserting iTunes catalog."
C_4	Change Behavior	.92	Proposed to change behavior to ameliorate problems in the scenario, *e.g.*, "Reduce the volume of music"; "Have phone screen facing up (when on table) so that it will be visible when it lights up during a call"

Table 9.2: Coding scheme for control suggestions. Participants' suggestions to improve Laκsa for each scenario were coded with values: 0=Ineffective, 0.5=Partially effective, 1=Effective. Each code may be counted multiple times depending on how many suggestions were made. Inter-coder reliabilities (κ) were calculated with a 50% random sample by a second coder.

9.5.4 PERCEPTION OF APPLICATION AND EXPLANATIONS

We were interested in how participants perceived Laκsa and its explanations for each scenario. As a manipulation check of the scenario designs, we asked participants how they perceived Laκsa's **Behavior Appropriateness** (7-point Likert scale: -3=Strongly Inappropriate, 3=Strongly Appropriate). We also asked if they agreed or disagreed that the explanations were useful in the scenario (**Explanation Usefulness**; 7-point: -3=Strongly Disagree, 3=Strongly Agree).

Next, we describe how participants used intelligibility, and how that impacted their understanding. We treated S1 as a warm up for participants and excluded its results from our analyses.

9.6 RESULTS

Using a local recruiting website, we recruited 19 participants (11 females) with ages 19 to 65 (Median=26) years. We dropped P13 because he did not continue beyond S2, and did not understand the scenarios well. Nine participants were graduate students, and three were undergraduates. P01, P16, and P17 were students in a computer-related field (Electrical and Computer Engineering, Software Engineering, and Learning Technology). P18 was a web programmer, while the others spanned a wide range of areas (*e.g.*, actor, pianist, field interviewer, hospital administrator, chemical engineering, retiree). 11 participants owned smart phones and one participant did not even own a cell phone. We engaged each participant for 1h 44min on average (range: 1h 29m to 1h 58m). Each participant was compensated $20.

While we strove to make all user experiences consistent for the experiment, we also strove to have Laκsa behave faithfully to the scenarios the participants were situated in, and what they did. Consequently, there was some variability in what Laκsa sensed and the resulting explanations. For example, location sensing accuracy depended on where the participant decided to walk to, weather conditions and other environmental factors affecting signal strength; when the participant walked to the café in S4, she may heard background music, or found a seat nearby people who are talking.

For S4, participants exhibited two distinct behaviors to explore the hypothetical situation: they just sat where they were and tried to use the What If explanation facility (S4-if, 10 cases), walked to the café to test Laκsa *in-situ* (S4-situ, 11 cases), or performed both activities. Hence, we treat these as distinct scenarios.

In this section, we report results of participants' perception of Laκsa's behavior and explanations, how they used intelligibility, their understanding of Laκsa's behavior, and how their usage affected their understanding. We supplement the quantitative data with descriptions of what participants did and said, and provide interpretations.

9.6.1 PERCEPTION OF APPLICATION BEHAVIOR AND EXPLANATIONS

A one-way ANOVA with Scenario as the factor found a difference in Behavior Appropriateness ($F_{3,25}$=3.90, p<.05), specifically, that Laκsa's behavior was perceived as more inappropriate for S2 and S3 than for S4 (contrast test: p<.01). This verifies the manipulation check that participants felt that Laκsa behaved inappropriately for S2 and S3 (see Figure 9.2, Left).

Overall, participants perceived explanations as useful (M=1.52, Std.Err.=0.23), even though perceived application behavior appropriateness varied (see Figure 9.2). A one-way ANOVA with Scenario as the factor found a difference in Explanation Helpfulness ($F_{3,25}$=3.90, p<.05), specifically, that explanations were less helpful in S2 than S4, and S3 indistinguishable with S2 or S4 (Tukey HSD test: p<.05; see Figure 9.2, Right).

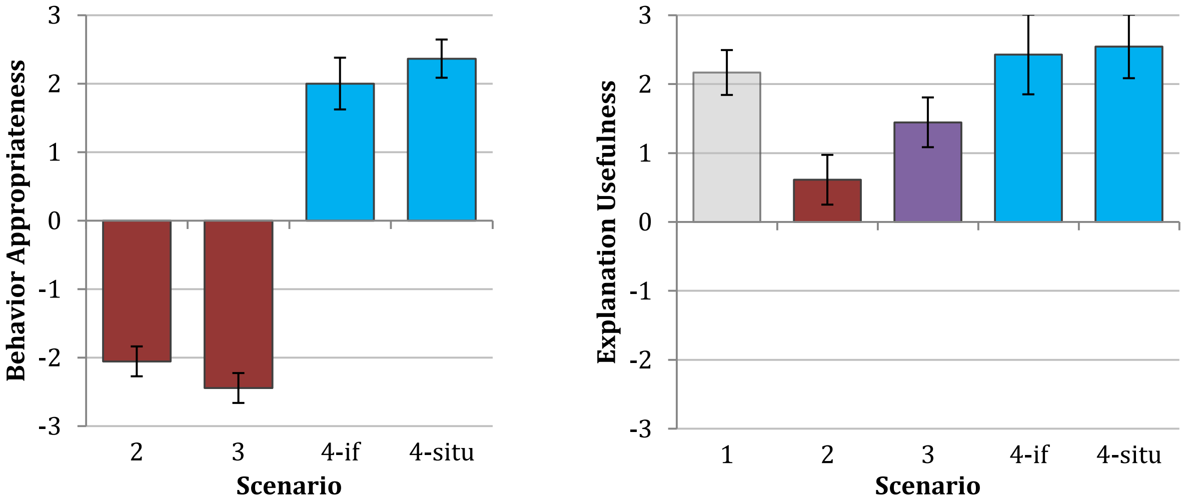


Figure 9.2. Perceived Application Behavior Appropriateness (Left) and perceived Explanation Usefulness (Right) across scenarios. We include S1 in the graph for comparison, but not in our analysis. Error bars indicate standard errors.

Next, we characterize how participants used intelligibility: how often they looked at explanations, and for how long.

9.6.2 INTELLIGIBILITY USAGE

From usage logs combined across S2 to S4, we determined participants' overall usage of intelligibility (see in Table 9.3), and their usage for each explanation type (see Table 9.4). Most participants actively looked at many Explanation Types (Median=8), many times (View Count Median=21), for about 3 minutes per scenario. This suggests they valued intelligibility enough to use it. They also tended to look more at deeper contexts (Place or Sound) than just Availability (Context Ratio Median=1.4). Some participants were very engaged in using intelligibility (View Count Max=65, Scenario Duration Max=12.5min), while some were conservative: min 2 views (P08S4-if), 1 explanation type (P14S4-situ), scenario duration <1 min (P08S4-if), or not looking at deeper contexts (Context Ratio=0, P02S2, 7 participants for S4-if, P05S4-situ, P14S4-situ).

From Table 9.4 Left, we identify which explanation types were more popular, *i.e.*, higher view count. The Availability What explanation was the first page that participants saw when they turned the screen on, so it has the highest count. Availability Inputs is also high because most participants used it as a gateway to see explanations of deeper contexts. Availability History was popular because participants had to ask about specific events in the past. Participants viewed Outputs to see the expected inferences that were not made (particularly for Availability, and Sound). Although participants seldom viewed Definitions, they did so more for Sound because they were less familiar with its concepts. Due to the temporal nature of Sound, 9 participants played audio clips of what Laκsa heard (Situation). 9 participants also used the Refresh function 30 times in total to get immediate feedback about the inference. They used it mostly during S3 and S4-situ, about Availability and Place, and for What, Inputs explanation types.

Per Scenario	Mean	Std. Dev.	Std. Error	Min	Median	Max
Steps (unfiltered)	27	16	2	2	23	71
View Count	24	15	2	2	21	65
# Explanation Types	7.9	3.4	0.4	1	8	19
Context Ratio	1.8	1.8	0.2	0	1.4	8
Total Duration (s)	205	136	18	52	196	749

Table 9.3. Summary statistics of intelligibility usage by participant scenario.

	Total View Count			Median Duration (s)		
	Availability	Place	Sound	Availability	Place	Sound
What + Certainty	232	114	69	5.7	3.1	3.2
History	130	5	3	7.5	-	-
Outputs + Certainty	84	102	33	6.4	5.6	4.9
Inputs	217	45	60	7.1	6.4	6.0
Why	26	16	44	3.5	9.9	6.1
Why Not (Why Alt)	21	35	41	4.7	9.6	4.9
What If	31	-	-	24.8	-	-
Definition	5	7	28	-	-	6.1
Situation	-	-	15			

Table 9.4. Usage of explanation types: total view count of explanation types for all participant scenarios, and median durations for respective views (for Total View Count > 15). Mean View Count per scenario can be calculated by dividing by number of participant scenarios, N=57. Colors show a heat map of values and relate to the numerical value.

We can also see how much time participants spent looking at each explanation type (Table 9.4, Right). While What If was not used often, when it was, participants spent significant time with it. This is because it is an interactive facility rather than a static display. For the other explanation types, participants on average spent less than 10 seconds viewing them, and may even view them as quickly as about 3 seconds. Furthermore, participants spent more time on explanation types that were more complex or contained more information (*e.g.*, Availability Inputs > Why, Place Why > Inputs, Sound Inputs > What).

9.6.3 CORRELATIONS BETWEEN INTELLIGIBILITY USAGE AND ITS IMPACT

Given the variation in intelligibility usage, we next explored how that affected participant understanding, control suggestions, and perception. We calculated correlations between our metrics of intelligibility usage, user understanding, control suggestions, and perception (see Table 9.5). These suggest some relationships, which we interpret. When participants perceived the application as behaving *less* appropriately, they viewed more explanations (a) and more types (b), spent more time exploring explanations (c), provided more suggestions for controlling and fixing the behavior (d), but perceived explanations as less useful (e). They had higher Understanding scores when they viewed more explanations (f, h), viewed more about deeper contexts than shallower ones (i), or spent more time looking at explanations (j). The same was true for their Control score, perhaps due to their improved understanding (k). Strangely, explanation usefulness was not correlated with intelligibility usage (g), and participants who perceived explanations as more useful had fewer suggestions for effective control (l).

	Usage				Impact		
Pearson's *R*	View Count	# Explanations	Context Ratio	Total Duration	Understanding	Control	Explanation Usefulness
App Behavior Appropriateness	-.37^{a}	-.32^{b}	-.02	-.41^{c}	-.11	-.34^{d}	.40^{e}
View Count		.81	.20	.63	.43^{f}	.29	-.17^{g}
# Explanation Types			.26	.45	.36^{h}	.30	-.15
Context Ratio				.06	.42^{i}	.30	-.05
Scenario Duration					.20^{j}	.13	-.08
Understanding						.41^{k}	.05
Control							-.23^{l}

Table 9.5. Correlations between usage of, and impact due to intelligibility. Significant correlations underlined (single: p<.05, double: p<.01). Superscript letters refer to interpretations in text passage.

Now that we see some potential relationships between intelligibility usage and its impact, let us explore how well participants understood Laκsa, and conduct statistical tests of whether and how usage affects understanding.

9.6.4 USER UNDERSTANDING AND CONTROL SUGGESTIONS

We first report the average understanding participants had. For each scenario, participants articulated 0 to 8 correct beliefs about Laκsa's behavior (Median=4). Figure 9.5 (Left) shows the distribution of their correct beliefs. 41% of the beliefs were about the awareness of the inferred Value for Place and Sound, 28% about a broader understanding of the inference (Alternative Values and Certainty), 15% about the Inputs state and Model mechanism, and 2.5% about deeper Technical details. 14% of the beliefs were drawn from the situation to justify Laκsa's behavior. While not coded, some participants expressed incorrect mental models, such as believing that the

Place inference influences Sound inference, and vice versa (*e.g.*, P14-S2: *"[Laκsa] infers location first [Office], then uses that to infer sound is likely talking [, instead of music]"*).

Participants provided 0 to 6 correct Control Suggestions (Median=2) for each scenario, and had an average Control Score of 2.10 (Std Err=0.29). This is significantly greater than 1 (*i.e.*, H_0: Score>1, p<.01). Figure 9.6 (Left) shows the distribution of effective and partial Control Suggestions to improve Laκsa's behavior: Availability Rules (29%), Settings (27% Place, 8% Sound), Behavior Change (36%).

Regardless of how participants used intelligibility, on average, they had non-zero Understanding and Control scores (Figure 9.5, Left; Figure 9.6, Left). However, the extent and pattern of intelligibility usage did affect how well participants understood Laκsa, as we shall see next.

9.6.4.1 IMPACT OF INTELLIGIBILITY USAGE ON UNDERSTANDING AND CONTROL SCORES

From the correlations between usage and impact (Table 9.5), we chose View Count and Context Ratio as factors of intelligibility usage. We split View Count into discrete intervals of 10 counts (sample size: 9-16); we split Context Ratio into two groups Shallower (N=34) and Deeper (N=20), where participants saw twice as many explanations about Place or Sound than Availability. We also consider Smartphone Ownership as a factor which may influence understanding and control.

IMPACT ON UNDERSTANDING SCORE

We performed a mixed-model analysis of variance with Participant as the random effect, nested in Scenario; View Count, Context Ratio, and Smartphone ownership as main effects; and Understanding Score as the dependent variable (R^2=.466). We found a marginal difference across View Count groups ($F_{4,45}$=2.54, p=.05; see Figure 9.3, Left), and a contrast test found that when View Count <30 instead of ≥30, Understanding Score was lower (p<.05). Moreover, the score was higher when participants viewed explanations of Deeper contexts ≥2 times more than Shallower ones ($F_{1,45}$=6.92, p<.05; see Figure 9.3, Right). There was no difference in Understanding across Smartphone Ownership.

IMPACT ON CONTROL SCORE

We performed a similar mixed-model analysis of variance for Control Score as the dependent variable (R^2=.466). There was no difference across View Count groups (p=n.s.), but Control Score was higher for Deeper context ratios than Shallower ones ($F_{1,45}$=8.00, p<.01) and higher for participants who owned smart phones ($F_{1,45}$=4.38, p<.01). See Figure 9.4, Right. To investigate whether Context Ratio or Smartphone Ownership better explains the variance in the Control Score, we fit three linear models varying the effects. Our results in Table 9.6 suggest that Context Ratio is a more predictive factor than Smartphone Ownership.

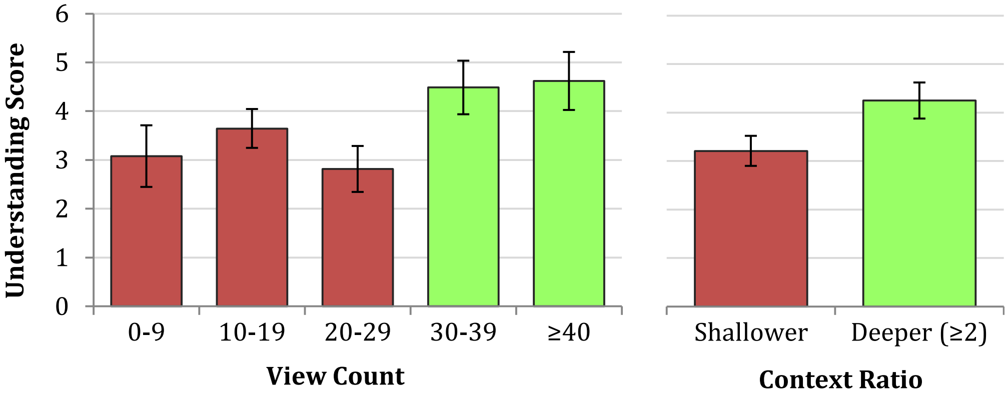


Figure 9.3. Participants had a higher Understanding Score when they viewed ≥30 explanations than fewer (p<.05, Left), and when they viewed explanations of Deeper contexts ≥2 times more than Shallower ones (p<.05, Right).

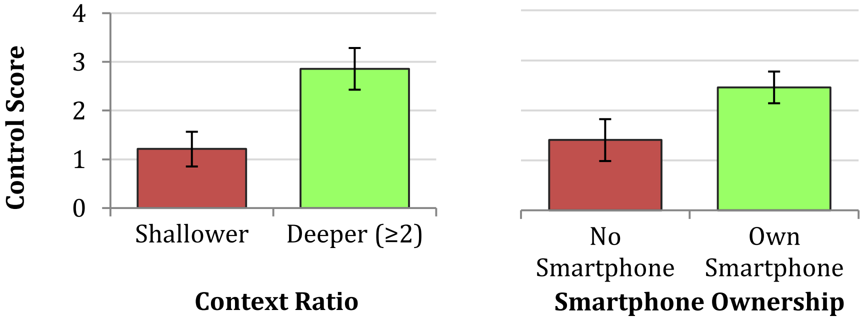


Figure 9.4. (Left) Control Score is higher when participants ask more explanations about Deeper contexts (Place and Sound) than Availability. (Right) Participants who owned smart phones also had higher Control scores. Control scores were not influenced by View Count.

	View Count	Context Ratio	Smartphone Ownership	R^2
1	✓	✓	✓	.466
2	✓	✓		.413
3	✓		✓	.380

Table 9.6. Linear regression models with R^2 values showing which factors better explain Control Scores.

Intelligibility usage also affected the depth of understanding participants expressed. Figure 9.5 (Right) shows that when participants had deeper Context Ratios, they described 2.0 times more details about the factor inference (Alternative Values, Certainty, Inputs, Model, and Technical), but mentioned fewer Situation Justifications. Similarly, they provided 2.2 times more types of Control Suggestions, especially about Settings (Figure 9.6, Right).

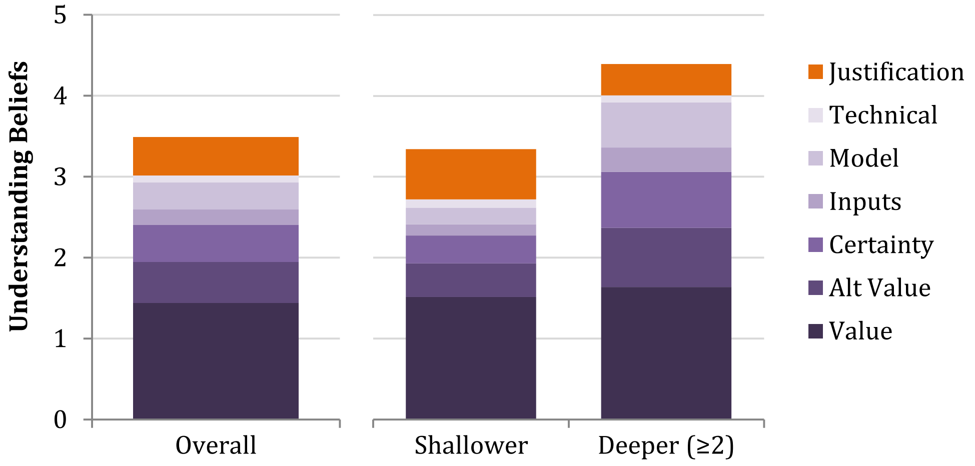


Figure 9.5. Distribution of belief types of Understanding overall, and by Context Ratio; normalized per scenario. Similar distribution for low View Count (<30) vs. high.

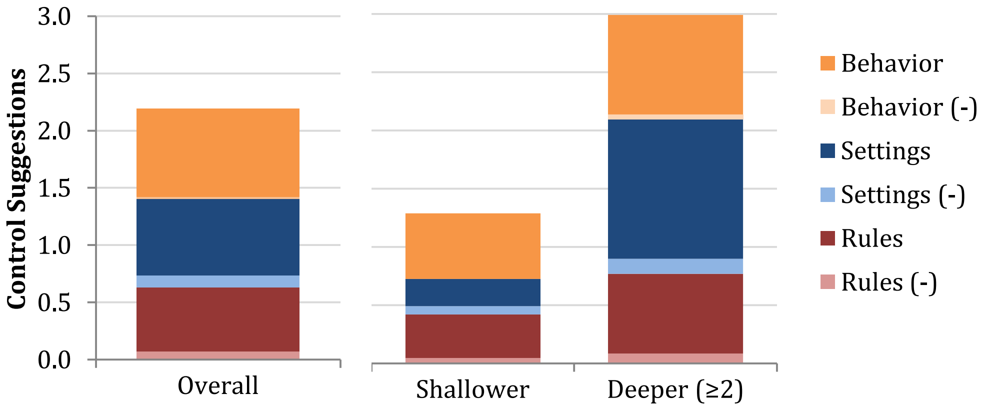


Figure 9.6. Distribution of effective suggestions participants made to improve Laκsa's behavior; normalized for each scenario. (-) denotes partially effective suggestions with side-effects. Note this is not the weighted Control Score.

In summary, our results show how participants were willing to use intelligibility, and how quickly or deeply they used it. This satisfies our hypothesis that more Intelligibility Usage (View Count and Context Ratio) improves Understanding.

9.7 Discussion, Implications, and Recommendations

We discuss what we have learned about how intelligibility is used, and how that affects user understanding of context-aware applications. These have implications on how intelligibility should be provided, and how we should design intelligibility to facilitate its more effective use.

9.7.1 Usage and Usefulness of Intelligibility

By the extent that our participants viewed the explanations, we can conclude that intelligibility was *useful* for them to (i) engage with intelligibility (some participants deeply so), (ii) rate explanations as useful, and (iii) gain better understanding of application behavior. We next discuss how they used intelligibility, and how certain usage patterns were more effective in improving user understanding.

9.7.1.1 Self-Reported Usefulness of Intelligibility

Similarly to our previous results in Chapter 8, perceived explanation usefulness was also affected by application behavior appropriateness (Section 9.6.1). Nevertheless, on average, participants perceived intelligibility explanations as useful even when the application behaved inappropriately (see Figure 9.2). Participants found the explanations interesting and viewed them to satisfy their curiosity (*e.g.*, P01S1: *"for figuring out what the sensed factors are like"*). In S4, P02 likes that the What If explanation *"enabled [him] to get a reading before even leaving [for the café]."* Explanations helped reinforce the appropriateness of application behavior by showing that the application also correctly inferred other details (*e.g.*, for S4, P01 was satisfied that the Sound was correctly inferred as Music and the Location was sensed correctly as the Café).

Even when the application behaved inappropriately, some participants found explanations useful. For S2, P10 found the explanations *"surely helpful"* and was able to determine that the music heard was recognized as talking, because the song *"reminds [him]of people talking."* For S3, P18 felt that the explanations were *"very plain"* and mentioned that *"I could tell why the problem was that it did not detect the location of the library."*

However, there were occasions when participants did not find explanations useful. For S1, P11 confessed that *"phones, in general, makes [her] cranky"* and felt that there was too much information and too many details. In S2, P06 gave a compromise rating for explanation usefulness (Neither Disagree Nor Agree), giving the reason: *"I guess I understood like what setting it was in, but [Laκsa]* *doesn't really seem to sense its environment correctly."* For S3, P07 *"was trying to figure out why it sends out at to the office opposed to library and was having difficulty figuring that out."*

Finally, although explanations may not be useful for certain situations, some participants felt that, in general, it is useful (*e.g.*, for S2, P08 felt that she would have *"naturally looked at the explanations,"* in general, and although, *"in this case [S2], it was like a waste of time, but I don't regret looking at it"*).

9.7.1.2 DIVERSE USAGE OF EXPLANATION TYPES

Participants used a diverse range of explanation types (see Table 9.4) and in diverse ways. What and Inputs were conduits to other explanations for participants to learn deeper reasons. However, although some explanation types were used less than others, participants viewed them for longer durations when they did (*e.g.*, Place Why / Alt). Furthermore, as with Section 7.8 [Lim and Dey, 2011a], the sequence diagrams of our participants revealed a variety of usage styles (*e.g.*, quick comparison between Why and Why Alt reasons, diving into a deeper context after going straight to Availability Inputs).

Unlike what was found in Section 4.6.3 [Lim, Dey, and Avrahami, 2009], our participants felt that the What If explanation was easy to use and liked it (*e.g.*, P11S1: *"[Using] it was just more fun … I like to think of hypothetical things, but it also gives me a sense of what the phone is capable of, and helps to develop trust when you know what to expect"*). In fact, for S4, 10 participants chose to ask What If instead of immediately walking to the café. However, this fascination with What If can also give users false trust since it obscures potential pitfalls in sensing. Participants who used What If in S4 may not realize how noisy the café may be or that the Place inference was not particularly good there. P11 did not bother to explore Laκsa's inference in-situ because *"technology is supposed to make your life easier; you shouldn't have to waste time to make sure it works right."* Perhaps providing warnings that sensing can fluctuate due to environmental conditions may help users be more careful when using What If.

While not explicitly an explanation, Refresh was used to understand what Laκsa was sensing and inferring in the moment. For S3, P12 and P17 anticipated Laκsa would not sense its location well at the Library, and refreshed the display to track the location sensing. They learned before arriving at the Library that Place and Availability were wrong. Similarly, for S4-situ, 7 participants refreshed to (impatiently) check if their status had been set to Available and/or Place as Café. In fact, because of this sluggishness, some participants attributed mis-inferences to lag.

Occasionally, participants forgot what had happened recently, *e.g.*, for S2, P07 thought he was talking to the experimenter at the time Laκsa inferred Sound as Talking. Had he played the recorded audio of that time (Situation), he would have learned that only singing was heard. Using the played audio, P15 and P16 were able to identify guitar sounds when Sound was finally recognized correctly as Music. Hence, in combination with History, Situation explanations can help jog a user's memory of what was happening, independent of the application's inference. This helps them form Situation Justifications for the application behavior. How may we also provide Situation explanations for contexts other than Sound? For Place, perhaps by showing a photograph at the location (if one was taken at the same time). For Motion recognition, perhaps by animating an *interpreted* diagram of how the phone was moving (derived from accelerometer data).

While our earlier research into intelligibility sought to prioritize providing some explanation types over others (Chapter 4 [Lim, Dey, and Avrahami, 2009] and Chapter 5 [Lim and Dey, 2009]), our more recent findings in Chapter 7 [Lim and Dey, 2011a], and these findings suggest instead to provide a diversity of explanation types will be helpful to support different learning and troubleshooting strategies users have.

9.7.1.3 DEEPER USAGE OF INTELLIGIBILITY

Our quantitative results indicate that viewing more explanations, especially about deeper contexts can lead to deeper understanding, and more effective control suggestions for improving the application behavior. So, to promote user understanding, we need to encourage users to dig for more explanations, and to dig deeper. Perhaps, if the user starts asking questions, the application could hypothesize faults, and highlight which factors are probably causing them. These guesses could come from a knowledge base of typical faults (Section 7.10.4), or be triggered when inferences Certainty becomes too low (*e.g.*, <80% as suggested in Section 8.12).

9.7.1.4 INTELLIGIBILITY AFFECTED BY FAMILIARITY WITH CONTEXT TYPE

Also observed in Section 8.9 [Lim and Dey, 2011b], our participants indicated less familiarity with Sound than they did with Place, and this affected the usage and usefulness of intelligibility. Participants had fewer Control Suggestions for Sound settings than for Place, despite higher View Counts for explanations about Sound than Place (Table 9.4, Left). This lack of familiarity also appears to influence their perception of Explanation Usefulness, where it was lower for S2 (Sound misinferred) than for S3 (Place misinferred), even though participants perceived the application behavior as equally poor in both scenarios. Perhaps providing easier access to Definitions and repeated exposures to the intelligibility features can promote familiarity, and allow users to gain more understanding of more novel contexts.

9.7.1.5 INTELLIGIBILITY FOR CONTROL

The lack of familiarity with Sound also hindered our participants' ability to provide Control suggestion to improve its inference. Only a few suggestions were made: *e.g.*, P10 suggested using her iTunes music library as a training dataset, and P09 suggested *"adjusting the levels for Periods of Silence for conversation vs. music."* Clearly, we should provide intelligibility to facilitate control (as recommended in Section 5.7 [Lim and Dey, 2009]), perhaps by just adding a Definitions-style explanation that describes control mechanisms and some consequences of adjusting them.

9.7.2 CONSTRAINTS FOR INTELLIGIBILITY

While the upper bounds of our participants' usage of intelligibility may give an indication of engagement, the lower bound may portend the limits to which some users may be willing to use intelligibility. Therefore, we derive some time and view constraints for intelligibility. Participants only spent about 3-10 seconds viewing each explanation, so each explanation page needs to be correctly and effectively interpreted within that short duration. Perhaps if an explanation cannot be understood within that duration, it should be split into multiple parts where the user can ask for more *on demand*. Furthermore, our quicker participants spared only about 1-3 minutes exploring explanations for each incident. This may be even shorter without the experimenter demand effect when users explore intelligibility outside of a user study. Hence, question asking should be streamlined to facilitate multiple views (~20) within about 2 minutes before the user gives up.

With our scenarios, we have focused on investigating the usage of intelligibility about incidences *in-situ* and in the moment (or shortly after), or a future hypothetical situation. However, users may postpone investigating an incident until they have more time. Under those circumstances, the time constraints for using intelligibility may not be so tight, but users may need more information to remind them what happened (*e.g.*, richer Situation and History explanations).

9.7.3 HOW MUCH INTELLIGIBILITY IS "GOOD ENOUGH?"

We have discussed how intelligibility can be beneficial to users by improving their understanding of application inference and behavior. We have also discussed how to amplify these benefits through deeper use of intelligibility. However, is this deeper use sufficient or excessive, *i.e.*, what are the upper and lower bounds for how much intelligibility to provide? Our results suggest that users may use intelligibility at least to the depth that was provided in Laκsa, and therefore it is not *excessive*. To decide how much intelligibility usage will be *sufficient* for users (in terms of benefit, independent of effort required), we can consider a model of how intelligibility influences understanding, trust, and control (see Figure 9.7). This study covered the path from Usage to Control (in black), while our previous work in Chapter 4 [Lim, Dey, and Avrahami, 2009], Chapter 5 [Lim and Dey, 2009], and Chapter 8 [Lim and Dey, 2011b] have also explored the link between Understanding and Trust. In particular, in Section 8.12, we recommend when and how much intelligibility to provide to help instead of harm user impression (Trust). From Figure 9.6 (Left), we can see that our participants gave at least one effective Control suggestion on average. If one change is sufficient to improve the application's performance, then this should be good enough. However, this depends on the type of suggestion (see Figure 9.6): changing a Rule is brittle and may not be a robust solution in the long-run; changing Behavior puts extra strain on the user's actions and habits, so they may not favor this repeatedly too; changing Settings is more robust and less tedious, but may suffer from side-effects (*e.g.*, adjusting weights in a machine learning system). Therefore, users may need more suggestions to be able to provide satisfactory control, which means that they should explore explanations of deeper contexts more than of the shallower application context.

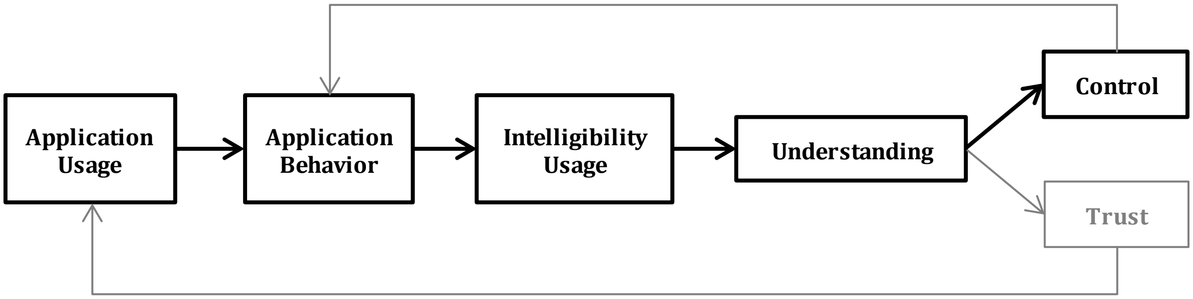


Figure 9.7. Model of influence indicating how the usage of Intelligibility may influence a user's ability to Control, sense of Trust, and usage of the context-aware application. Lines in grey were not explicitly explored in this study.

9.8 LIMITATIONS AND FURTHER WORK

While our quasi-field study with an interactive prototype and realistic scenarios can provide insight into how users use and benefit from intelligibility, there are limitations due to its controlled set-up and brief duration. Our study covered only a handful of situations where intelligibility is useful, but we expect more situations and even *unanticipated* ones as users use Laκsa in their daily lives. Our study presented a bias by selecting scenarios where participants are more motivated to use intelligibility to troubleshoot the application behavior. Our purpose was to demonstrate that there exist situations where users will and do use intelligibility. Under everyday usage, we expect there to be many situations where users will not have an urgent need to use intelligibility, and may either forego or postpone its use.

Furthermore, participants had only two hours to familiarize themselves with the UI, and go through four scenarios. As such, their experience only covered the *initial transient* usage of intelligibility, as novices. We expect their usage patterns and knowledge of the application and its inference to evolve as they use intelligibility over time. Therefore, for future work, we plan to deploy Laκsa in the field and over a few weeks to overcome the aforementioned limitations. We intend to study how prolonged use of intelligibility impacts long-term understanding and trust of context-awareness.

In order to focus on the usage of intelligibility in a free-form manner, we provided intelligibility *on demand* instead of *always on*. Moreover, we did not include a baseline control condition where participants did not see any intelligibility features, or saw a limited set of explanations. This can be valuable to for evaluating the impact of intelligibility by comparing intelligible and non-intelligible versions and also help control for personality traits that may have coupled a user's ability to understand Laksa2 and her propensity for using intelligibility more. Nevertheless, we refer to our previous controlled studies in Chapters 4 and 8 that had explored the impact of providing intelligibility on understanding and trust.

In this study, we have investigated the usage and impact of intelligibility in one context-aware application. Although this is one data point in a sparse design space, Laksa2 was designed to cover a range of contexts (Availability, Place, Sound) and inference models (Rules, Decision Tree, Naïve Bayes), so it spans many features of context-aware applications. This serves to increase the generality of our results. While Laksa2's UI presentation was optimized for usability, we architected Laksa2's structure to be generalizable *depth-wise* from a multi-level context hierarchy (layering of contexts as inputs and outputs) and *breadth-wise* by spanning explanation question types.

9.9 CONCLUSIONS

We have presented a quasi-field study where we measured how participants naturalistically used an intelligible context-aware application in scenarios representing real-world, "everyday" situations. We investigated how that usage affects their understanding of the application behavior. The application was an iteration over the Laκsa prototype with more streamlined and usable intelligibility features. We found that viewing more explanations, especially more about deeper contexts can further improve user understanding of application inference in our experimental scenarios. We provided potential implications for promoting more effective intelligibility usage, time constraints within which users are willing to view intelligibility, and discussed how much intelligibility should be provided to sufficiently improve user understanding of context-aware applications.

10 DISCUSSION & CONCLUSION

This thesis has investigated how to provide intelligibility in context-aware applications and shown that intelligibility can improve end-user understanding and trust in these adaptive applications. As we conclude, we summarize the contributions made in this thesis, and discuss avenues for future work.

10.1 SUMMARY OF CONTRIBUTIONS

We summarize our contributions in terms of the three high-level stages of the thesis approach: (i) requirements for intelligibility, (ii) support for intelligibility, and (iii) evaluation of intelligibility in context-aware applications.

10.1.1 TAXONOMY OF EXPLANATION TYPES FOR INTELLIGIBILITY

In Chapter 5, we elicited a range of question types end-users want to ask of context-aware systems: What, What Else, Certainty, Why, Why Not, What If, How To, Inputs, Outputs, Control, and Situation. This forms the basis of the taxonomy of explanation types we use to provide intelligibility in context-aware applications. We believe that providing these explanation types will improve user satisfaction and acceptance of context-aware applications.

In Chapter 3, we summarized all the explanation types that we have explored from our elicitation study (Chapter 5), and from subsequent design exercises with higher fidelity intelligible prototypes (Chapters 7 and 9). These additional explanation types include: When, History, and Description.

10.1.2 IMPLEMENTATION SUPPORT FOR INTELLIGIBILITY

In Chapter 6, we described the Intelligibility Toolkit, which we have developed to make it easier for developers to provide many explanation types in their context-aware applications. The toolkit provides automatic generation of 12 explanation types for at least 10 popular inference models in context-aware applications. It consists of *structural components*:

- **Queries** to encapsulate questions about inferred contexts, and
- **Explanation Expressions** to represent explanations in data structures

and *function components*:

- **Explainers** to generate explanations,
- **Reducers** to simplify complex explanations,
- **Presenters** to render explanations for end-users to consume,
- **Queriers** to present interfaces for end-users to ask questions, and
- **Selectors** for contextually select queries or reducers to provide.

Explanations can be provided as either *rule traces* or *weights of evidence*. The toolkit is extensible to support new explanation types, model types, reduction heuristics, and presentation formats.

10.1.3 DESIGN RECOMMENDATIONS FOR INTELLIGIBILITY

In Chapter 5, we identified which explanation types to provide to end-users and under which circumstances (Application Behavior Inappropriateness, Situation Criticality, Application Goal-Supportiveness, Recommendation Role, Number of Externalities) to best provide them. This is encoded in a table of design recommendations (see Table 5.6) along with recommended provision mechanisms (see Table 5.7). These findings help to inform us about context-sensitive intelligibility.

In Chapter 7, through design iteration and a usability study of Laκsa, an intelligible mobile context-aware application, we developed several design principles for usable intelligibility. We investigated the use of intelligibility for the mobile contexts of Availability, Place, Motion, and Sound activity. Our findings emphasize

- Visualizing explanations to allow quick interpretation, but also including textual explanations to scaffold the graphics.
- Making explanations usable and quickly consumable (by reducing explanation volume or detail, and aggregating explanations).
- Focusing on providing explanations that facilitate user control, and excluding explanation details that do not allow users to improve or change the application behavior.

- Streamlining questioning to reduce the number of options users have when seeking explanations.
- Using terminology and descriptions more related to the real-world activity than technical aspects of the application inference.
- Integrating domain knowledge in explanations (*e.g.*, distinguish indoor and outdoor location sensing via Wi-Fi and GPS respectively).
- Supporting effective problem solving and debugging strategies (so that users can quickly understand the application issues before giving up).

In Chapter 8, drawing from our evaluation of the helpfulness and harmfulness of intelligibility across an application certainty threshold of about 80-90%, we provided recommendations on when and how to provide explanations (see Table 8.9):

- No intelligibility if the overall application certainty and accuracy is low (below the threshold).
- Provide intelligibility *automatically* if overall application certainty and accuracy is high (above the threshold).
- Provide intelligibility *on demand* to allow users to ask for explanations if the application behaves inappropriately.

In Chapter 9, we draw some insights into design constraints for providing intelligibility for mobile context-aware applications, from our evaluation of Laκsa2:

- Enable users to acquire sufficient understanding with no more than *20 explanation views* per incident or session of use and within *2 minutes*.
- Enable users to interpret each explanation view in within *3-10 seconds*.

10.1.4 EVALUATIONS OF INTELLIGIBILITY

In Chapter 4, we found that providing reasoning trace explanations for context-aware applications to novice users, and in particular Why and Why Not explanations, can improve users' understanding and trust in the system. We also found that the complexity of How To and What If explanations may have impeded their usefulness and efficacy. We therefore sought to improve the usability and ease of understanding of How To and What If explanations to improve their usefulness.

In Chapter 8, we showed that intelligibility can positively or negatively impact user impression, depending on the application's certainty and behavior appropriateness. Intelligibility is helpful for applications with high certainty, but it is harmful for applications with low certainty, leading to the user losing even more trust in its capability. Nevertheless, intelligibility can help users to appreciate and forgive applications if they behave inappropriately and have low certainty.

In Chapter 9, through a quasi-field experiment with Laκsa2, we showed how usage of intelligibility affects user understanding of its behavior. We found that, in our experimental scenarios, users were willing to use intelligibility in Laκsa2, and that viewing more explanations, especially more about deeper contexts can further improve user understanding of application inference. From our results, we provided implications for promoting more effective intelligibility usage, time constraints within which users are willing to view intelligibility, and insights for how much intelligibility should be provided to sufficiently improve user understanding of context-aware applications.

10.2 LIMITATIONS

With the paucity of intelligible context-aware applications, and in order to conduct controlled user studies, we conducted most of our user studies with online questionnaires. This format allows us to collect a large amount of data and user perspectives regarding intelligibility. However, this approach suffers from a lack of realism where users can experience the intelligibility features over time and subtle effects and issues become manifested. Nevertheless, we have progressively shifted our investigation from abstract context-aware applications, to richly described applications, to a fully functional intelligible prototype.

10.3 ADDITIONAL RESEARCH OPPORTUNITIES

Through this dissertation work, we have begun to orientate our research of intelligibility from the lab to the real world. This introduces many issues and factors that will interact with intelligibility, and opportunities to investigate them. We describe a few future research opportunities.

10.3.1 INTELLIGIBILITY FOR CONTROL

Even though end-users of context-aware applications are expected to let the application function seamlessly without direct user input, users still desire a sense of control (*e.g.*, [Barkhuus and Dey, 2003; Bellotti and Edwards, 2001; Dey and Newberger, 2009]). They will occasionally need to control and configure the application to override its decision, or improve its performance for future situations. We hypothesize that, with the increased understanding due to intelligibility, users will more effectively control the context-aware application. By more *effective* control, we mean that the user will know *which parameters* to adjust to improve the application's inference and behavior, and that he can also change parameter *values* such that its inference for the current and, possibly, future situations will be improved.

With appropriate control interfaces and learning algorithms, we can extend our study on the usage of intelligibility to measure the impact of user control on application accuracy and performance. One way to control rule-based context-aware applications is through selected *Enactor Parameters* exposed by the Enactor framework [Dey and Newberger, 2009]. However, it is more challenging to control machine learning-based applications, since manual editing of the learned model may not guarantee improved performance or accuracy. Early work by Wong *et al.* [2011] shows promising results that certain algorithms can improve accuracy with feedback from end-users.

10.3.2 Social Intelligibility

Along with single-user applications, context-awareness is also being used for social applications (*e.g.*, [Lim and Dey, 2011a; Rosenthal, Dey, and Veloso, 2011]). Since providing intelligible contexts is useful to users of single-user applications, it is likely to also be useful for users learning about their social contacts or other relationships. However, intelligibility seeks to illuminate deeper levels of context information and reasoning. While this can help users to better understand their counterparts, the owner of this information may be less willing to share the information with others. Future work can explore the trade-off between the need for intelligible contexts in a social setting and the desire for privacy which will hinder providing such information. With such results, we can recommend intelligible context information that users feel comfortable sharing and have sufficient need to consume.

10.3.3 Field Study of Intelligibility

An important next step to validate our findings of the use and usefulness of intelligibility is to conduct a field study of an intelligible application. This will allow us to investigate how real-world concerns affect the use and impact of intelligibility, and how these measures change over time. By allowing users to freely use the application in their everyday lives, we also minimize any experimenter demand effects which may bias users to use intelligibility. A longitudinal study will also reduce the novelty effect of the early usage of intelligibility. We can also explore whether intelligibility remains useful after users have learned and become familiar with how the application inference works, and observe how the need for intelligibility changes over time.

As an early field exploration, we have conducted a pilot study in 2009 of IM Autostatus over 3-4 weeks [Lim and Dey, 2012a]. IM Autostatus is an intelligible instant messaging plugin that predicts when a buddy will respond to the user's message. We compared intelligible/non-intelligible versions with high/low accuracies (~60/80%). We found that users receiving intelligibility agree more with the application predictions, and had more detailed and correct mental models of the application behavior. We also determined that users used intelligibility (on demand), but this declined after a few days, especially for versions with low accuracy.

With further engineering of the Laκsa2 prototype, we can deploy the application to the app marketplace. This will allow us to explore intelligibility with a large pool of users over a longer period of time.

BIBLIOGRAPHY

Aamodt, A. (2004). Knowledge-intensive case-based reasoning in CREEK. In *Proceedings of the 7th European Conference on Advances in case-based reasoning (ECCBR 2004)*, 1-15.

Abowd, G.D., Atkeson, C.G., Hong, J., Long, S., Kooper, R. and Pinkerton, M. (1997). Cyberguide: a mobile context-aware tour guide. *Wireless Networks* 3(5), 421-433.

Abowd, G.D., Mynatt, E.D., and Rodden, T. (2002). The Human Experience. *IEEE Pervasive Computing* 1(1), 48-57.

Ahmed, S. (2011). Truth Table Solver. http://truthtablesolve.sourceforge.net/. Retrieved 1 March 2012.

Albinali, F., Intille, S., Haskell, W., and Rosenberger, M. (2010). Using wearable activity type detection to improve physical activity energy expenditure estimation. In *Proceedings of the 12th ACM international conference on Ubiquitous computing (Ubicomp '10)*. ACM, New York, NY, USA, 311-320.

Amazon Mechanical Turk. http://www.mturk.com. Retrieved 26 Jan 2012.

Ambient Devices. http://www.ambientdevices.com/products/what-is-ambient. Retrieved 3 April 2012.

Amft, O. and Tröster, G. (2008). Recognition of dietary activity events using on-body sensors. *Artificial Intelligence in Medicine*, 42(2), 121-136.

Anderson, J. R., Corbett, A. T., Koedinger, K. R., and Pelletier, R. (1995). Cognitive Tutors: Lessons Learned. *Journal of the Learning Sciences*, 4(2), 167-207.

Andrews, R., Diederich, J. and Tickle, A. (1995). A Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks. *Knowledge Based Systems*, 8, 373–389.

Antifakos, S., Schwaninger, A, and Schiele, B. (2004). Evaluating the Effects of Displaying Uncertainty in Context-Aware Applications. In *Proceedings of the 6th international Conference on Ubiquitous Computing (Tokyo, Japan, September 11 - 14, 2004). Ubicomp '04*. ACM, New York, NY, 54-69.

Antifakos, S. Kern, N., Schiele, B. and Schwaninger, A. (2005). Towards improving trust in context-aware systems by displaying system confidence. In *Proceedings of the 7th international conference on Human computer interaction with mobile devices & services (MobileHCI '05)*. ACM, New York, NY, USA, 9-14.

Assad, M., Carmichael, D.J., Kay, J. and Kummerfeld, B. (2007). PersonisAD: Distributed, Active, Scrutable Model Framework for Context-Aware Services. In *Proceedings of the 5th international conference on Pervasive computing (Pervasive'07)*, Anthony LaMarca, Marc Langheinrich, and Khai N. Truong (Eds.). Springer-Verlag, Berlin, Heidelberg, 55-72.

Avrahami, D. and Hudson, S.E. (2006). Responsiveness in Instant Messaging: Predictive Models Supporting Inter-Personal Communication. In *Proceedings of the SIGCHI conference on Human Factors in computing systems (CHI '06)*, Rebecca Grinter, Thomas Rodden, Paul Aoki, Ed Cutrell, Robin Jeffries, and Gary Olson (Eds.). ACM, New York, NY, USA, 731-740.

Bao, L. and Intille, S.S. (2004). Activity recognition from user-annotated acceleration data. In Proceedings of the Second International Conference on Pervasive Computing (Pervasive'04), 1-17.

Bardram, J. E. (2005). The Java Context Awareness Framework (JCAF) – A Service Infrastructure and Programming Framework for Context-Aware Applications. In *Proceedings of the 3th international conference on Pervasive computing (Pervasive'05)*, 98-115.

Barkhuus, L. and Dey, A.K. (2003a). Location-based services for mobile telephony: a study of users' privacy concerns. In *Proceedings of Interact 2003, 9th IFIP TC13 International Conference on Human-Computer Interaction.* Zurich, Switzerland: ACM Press, 709-712.

Barkhuus, L. and Dey, A.K. (2003b). Is context-aware computing taking control away from the user? Three levels of interactivity examined. In *Proceedings of the 5th international Conference on Ubiquitous Computing (Ubicomp '03)*. ACM, New York, NY, 149–156.

Barua, D., Kay, J., Kummerfeld, B. (2011). Personis-LF: user modelling framework with user controlled forgetting for pervasive lifelong personalisation. In *1st International Workshop on Intelligibility and Control in Pervasive Computing at Pervasive '11*.

Bicocchi, N., Mamei, M., and Zambonelli, F. (2010). Detecting activities from body-worn accelerometers via instance-based algorithms. *Pervasive Mobile Computing*, 6(4), 482-495.

Bellotti, V. and Edwards, W.K. (2001). Intelligibility and Accountability: Human Considerations in Context-Aware Systems. *Human-Computer Interaction*, 16(2-4): 193-212.

Bellotti, V., Back, M., Edwards, W.K., Grinter, R.E., Henderson, A., and Lopes, C. (2002). Making sense of sensing: Five questions for designers and researchers. In *Proceedings of the 20th international Conference on Human Factors in Computing Systems (CHI '02)*. ACM, New York, NY, 415-422.

Bicocchi, N., Mamei, M., and Zambonelli, F. (2010). Detecting activities from body-worn accelerometers via instance-based algorithms. *Pervasive Mobile Computing*, 6(4), 482-495.

Blair, C.E., Jeroslow, R.G., and Lowe, J.K. (1986). Some results and experiments in programming techniques for propositional logic. In *Journal of Computers and Operations Research*, 13, 5 (May 1986), 633-645.

Borriello, G., Brunette, W., Hall, M., Hartung, C., and Tangney, C. (2004). Reminding About Tagged Objects Using Passive RFIDs. In *Proceedings of the 6th international conference on Ubiquitous computing (UbiComp '04)*, 36-53.

Breiman, L. (1996). Bagging predictors. *Macine. Learning*, 24, 2 (August 1996), 123-140.

Breiman, L. (2001). Random Forests. *Macine. Learning*, 45, 1 (October 2001), 5-32.

Brunk, C., Kelly, J., and Kohavi, R. (1997). MineSet: an integrated system for data mining. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 135-138.

Bucur, D. (2011). Intelligible TinyOS sensor systems: explanations for embedded software. In *Proceedings of the 7th international and interdisciplinary conference on Modeling and using context (CONTEXT'11)*, Michael Beigl, Hedda R. Schmidtke, Henning Christiansen, Thomas R.

Roth-Berghofer, and Anders Kofod-Petersen (Eds.). Springer-Verlag, Berlin, Heidelberg, 54-66.

Bulling, A., Ward, J. A., Gellersen, H., and Tröster, G. (2008). Robust Recognition of Reading Activity in Transit Using Wearable Electrooculography. In *Proceedings of the 6th International Conference on Pervasive Computing (Pervasive '08)*, Jadwiga Indulska, Donald J. Patterson, Tom Rodden, and Max Ott (Eds.). Springer-Verlag, Berlin, Heidelberg, 19-37.

Bulling, A., Ward, J.A., Gellersen, H., and Tröster, G. (2009). Eye movement analysis for activity recognition. In *Proceedings of the 11th international conference on Ubiquitous computing (Ubicomp '09)*. ACM, New York, NY, USA, 41-50.

Bulling, A. and Roggen, D. (2011). Recognition of visual memory recall processes using eye movement analysis. In *Proceedings of the 13th international conference on Ubiquitous computing (UbiComp '11)*. ACM, New York, NY, USA, 455-464.

Byrne, R.M.J. and Johnson-Laird, P. N. (1992). The Spontaneous Use of Propositional Connectives. In *Quarterly Journal of Experimental Psychology Section A: Human Experimental Psychology*, 1464-0740, 45 (1), 89–110.

Cassens, J., and Kofod-Petersen, A. (2007). Explanations and Case-Based Reasoning in Ambient Intelligent Systems. In *Proceedings of the 2nd International Workshop on Case Based Reasoning and Context-Awareness*, Co-located with the 7th International Conference on Case Based Reasoning (ICCBR), vol. 271.

Cassens, J. (2008). Explanation Awareness and Ambient Intelligence as Social Technologies. *Thesis for the degree doctor scientiarum*, Norwegian University of Science and Technology, Trondheim, Norway, May 2008.

Chang, K., Hightower, J., and Kveton, B. (2009). Inferring Identity Using Accelerometers in Television Remote Controls. In *Proceedings of the 7th International Conference on Pervasive Computing (Pervasive '09)*, Hideyuki Tokuda, Michael Beigl, Adrian Friday, A. J. Brush, and Yoshito Tobe (Eds.). Springer-Verlag, Berlin, Heidelberg, 151-167.

Chang, K., Chen, M. Y., and Canny, J. (2007). Tracking Free-Weight Exercises. In *Proceedings of the 9th international conference on Ubiquitous computing (UbiComp '07)*, John Krumm, Gregory

D. Abowd, Aruna Seneviratne, and Thomas Strang (Eds.). Springer-Verlag, Berlin, Heidelberg, 19-37.

Cheverst, K. Davies, N. Mitchell, K., Friday, A. and Efstratiou, C. (2000). Developing a context-aware electronic tourist guide: some issues and experiences. In *Proceedings of the 18th SIGCHI conference on Human factors in computing systems (CHI '00)*. ACM, New York, NY, USA, 17-24.

Cheverst, K., Byun, H.E., Fitton, D., Sas, C., Kray, C., and Villar, N. (2005). Exploring issues of user model transparency and proactive behavior in an office environment control system. *User Modeling and User-Adapted Interaction* 15(3-4), 235-273.

Circular Error Probable (CEP), *Air Force Operational Test and Evaluation Center Technical*. Paper 6, Version 2, July 1987, p. 1.

Clancey, W. J. (1983). The Epistemology of a Rule-based Expert System: A Framework for Explanation. Artificial Intelligence, 20(3), pp. 215-251.

Chalmers, M. and MacColl, I. (2003). Seamful and seamless design in ubiquitous computing. In *Workshop: At the Crossroads: The Interaction of HCI and Systems Issues, Ubicomp '03*.

Cohn, G., Gupta, S., Froehlich, J., Larson, E., and Patel, S.N. (2010). GasSense: appliance-level, single-point sensing of gas activity in the home. In *Proceedings of the 8th international conference on Pervasive Computing (Pervasive'10)*, Patrik Floréen, Antonio Krüger, and Mirjana Spasojevic (Eds.). Springer-Verlag, Berlin, Heidelberg, 265-282.

Cohn, G., Morris, D., Patel, S.N., and Tan, D.S. (2011). Your noise is my command: sensing gestures using the body as an antenna. In *Proceedings of the 2011 annual conference on Human factors in computing systems (CHI '11)*. ACM, New York, NY, USA, 791-800.

Consolvo, S., McDonald, D.W., Toscos, T., Chen, M.Y., Froehlich, J., Harrison, B., Klasnja, P. LaMarca, A., LeGrand, L., Libby, R., Smith, I., and Landay, J.A. (2008). Activity sensing in the wild: a field trial of ubifit garden. In *Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems (CHI '08)*. ACM, New York, NY, USA, 1797-1806.

Conway, M., Audia, S., Burnette, T., Cosgrove, D., and Christiansen, K. (2000). Alice: lessons learned from building a 3D system for novices. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '00)*. ACM, New York, NY, USA, 486-493.

Crama, Y. and Hammer, P.L. (2011). *Boolean Functions: Theory, Algorithms, and Applications*. Cambridge University Press, New York, NY, USA.

Cramer, H., Evers, V., van Someren, M., Ramlal, S., Rutledge, L., Stash, N., Aroyo, L., and Wielinga, B. (2008). The effects of transparency on trust and acceptance in interaction with a content-based art recommender. *User Modeling and User-Adapted Interaction*, 18 (5): 455–496.

Cramer, H., Evers, V., van Someren, M., Ramlal, S., Rutledge, L., Stash, N., Aroyo, L., and Wielinga, B. (2008). The effects of transparency on perceived and actual competence of a content-based recommender. In *Proceedings of the Semantic Web User Interaction Workshop at CHI'08*, Florence, Italy.

Crittenden, J. and Evans, Parker. (2008). Speaker Recognition. *ECE 576 Final Project, Cornell University*. https://instruct1.cit.cornell.edu/courses/ece576/FinalProjects/f2008/pae26_jsc59/pae26_jsc59/. Retrieved 1 May 2012.

Davidoff, S., Ziebart, B.D., Zimmerman, J., and Dey, A.K. (2011). Learning patterns of pick-ups and drop-offs to support busy family coordination. In *Proceedings of the 2011 annual conference on Human factors in computing systems (CHI '11)*. ACM, New York, NY, USA, 1175-1184.

Damer, T.E. (2009). *Attacking Faulty Reasoning: A Practical Guide to Fallacy-Free Arguments* (6th. ed.). Belmont, CA. Wadsworth Cengage Learning.

Davis, R., Buchanan, B., and Shortliffe, E. (1977). Production rules as a representation for a knowledge-based consultation program. *Artificial Intelligence*, 8(1): 15–45.

Dearman, D., Varshavsky, A., Lara, E.D., and Truong, K.N. (2007). An exploration of location error estimation. In *Proceedings of the 9th international conference on Ubiquitous computing (UbiComp '07)*, John Krumm, Gregory D. Abowd, Aruna Seneviratne, and Thomas Strang (Eds.). Springer-Verlag, Berlin, Heidelberg, 181-198.

Dey, A.K. (2000). *Providing Architectural Support for Building Context-Aware Applications*, Ph.D. thesis, December 2000, College of Computing, Georgia Institute of Technology.

Dey, A.K. and Abowd, G.D. (2000). CybreMinder: A Context-Aware System for Supporting Reminders. In *Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing (HUC '00)*, Peter J. Thomas and Hans-Werner Gellersen (Eds.). Springer-Verlag, London, UK, 172-186.

Dey, A.K., Abowd, G.D., and Salber, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2–4): 97–166.

Dey, A. K. and de Guzman, E. (2006). From awareness to connectedness: the design and deployment of presence displays. In *Proceedings of the SIGCHI conference on Human Factors in computing systems (CHI '06)*, Rebecca Grinter, Thomas Rodden, Paul Aoki, Ed Cutrell, Robin Jeffries, and Gary Olson (Eds.). ACM, New York, NY, USA, 899-908.

Dey A.K., Mankoff, J., Abowd, G., and Carter, S. (2002). Distributed mediation of ambiguous context in aware environments. In *Proceedings of the 15th annual ACM symposium on User interface software and technology (UIST '02)*. ACM, New York, NY, USA, 121-130.

Dey, A.K. and Newberger, A. (2009). Support for context-aware intelligibility and control. In *Proceedings of the 27th international conference on Human factors in computing systems (CHI '09)*. ACM, New York, NY, USA, 859-868.

Dey, A.K., Sohn, T., Streng, S., and Kodama, J (2006). iCAP: Interactive Prototyping of Context-Aware Applications. In *Proceedings of the 4th international conference on Pervasive Computing (PERVASIVE'06)*, Kenneth P. Fishkin, Bernt Schiele, Paddy Nixon, and Aaron Quigley (Eds.). Springer-Verlag, Berlin, Heidelberg, 254-271.

Dourish, P. (2004). What we talk about when we talk about context. *Personal Ubiquitous Computing*, 8(1), 19-30.

Dourish, P., Adler, A. and Smith, B.C. (1996). Organising User Interfaces Around Reflective Accounts. *Reflection '96*, 235-244.

Dzindolet, M., Peterson, S., Pomranky, S. Pierce, L., and Beck, H. (2003). The role of trust in automation reliance. *International Journal of Human-Computer Studies*, 58(6): 697-718.

Edwards, W.K., Newman, M.W., and Poole, E.S. (2010). The infrastructure problem in HCI. In *Proceedings of the 28th international conference on Human factors in computing systems (CHI '10)*. ACM, New York, NY, USA, 423-432.

Everett, A. M. (1994). An Empirical Investigation of the Effect of Variations in Expert System Explanation Presentation on Users' Acquisition of Expertise and Perceptions of the System. *Unpublished Doctoral Dissertation*, University of Nebraska.

Fatcow "Farm-Fresh Web Icons". http://www.fatcow.com/free-icons. Retrieved 1 February 2012.

Fogarty, J., Hudson, S.E., Atkeson, C.G., Avrahami, D., Forlizzi, J., Kiesler, S., Lee, J.C., and Yang, J. (2005). Predicting human interruptibility with sensors. *ACM Transactions on Computer-Human Interaction* 12(1), 119-146.

Fogarty, J. and Hudson, S. E. (2007). Toolkit support for developing and deploying sensor-based statistical models of human situations. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '07)*. ACM, New York, NY, USA, 135-144.

Fong, J. (2010). Intelligibility and user control of context-aware application behaviours. In *Proceedings of the 7th International Conference on Pervasive Services (ICPS2010)*, 174-179.

Fong, J., Indulska, J., Robinson, R. (2011). A Preference Modelling Approach to Support Intelligibility in Pervasive Applications. In *Proceedings of the 2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), 8th IEEE Workshop on Context Modeling and Reasoning (CoMoRea 2011)*, 409-414.

Franois, J.M. (2010). Jahmm: An implementation of Hidden Markov Models in Java. http://code.google.com/p/jahmm/. Retrieved 9 Mar 2010.

Freund, Y. and Schapire, R.E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the International Conference on Machine Learning,* 148-156.

Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2), 337-374.

Froehlich, J., Dillahunt, T., Klasnja, P., Mankoff, J., Consolvo, S., Harrison, B., and Landay, J. A. (2009). UbiGreen: investigating a mobile tool for tracking and supporting green transportation habits. In *Proceedings of the 27th international conference on Human factors in computing systems (CHI '09)*. ACM, New York, NY, USA, 1043-1052.

Froehlich, J.E., Larson, E., Campbell, T., Haggerty, C., Fogarty, J., and Patel, S.N. (2009). HydroSense: infrastructure-mediated single-point sensing of whole-home water activity. In *Proceedings of the 11th international conference on Ubiquitous computing (Ubicomp '09)*. ACM, New York, NY, USA, 235-244.

Fukui, R., Watanabe, M., Gyota, T., Shimosaka, M., and Sato, T. (2011). Hand shape classification with a wrist contour sensor: development of a prototype device. In *Proceedings of the 13th international conference on Ubiquitous computing (UbiComp '11)*. ACM, New York, NY, USA, 311-314.

Gibson, J.J. (1979). *The Ecological Approach to Visual Perception*. Boston: Houghton Mifflin.

Glass, A., McGuinness, D., and Wolverton, M. (2008). Toward establishing trust in adaptive agents. Proc. IUI'08. 227-236.

Google Web Toolkit. http://code.google.com/webtoolkit/. Retrieved 16 September 2008.

Graesser, A.C., Person, N., Huber, J. (1992). Mechanisms that generate questions. In: Lauer, T.W., Peacock, E., Graesser, A.C. (Eds.), *Questions and Information Systems*. Lawrence Erlbaum, Hillsdale, NJ, pp. 167–187.

Graesser, A. C., Langston, M. C. and Baggett, W. B. (1993). Exploring information about concepts by asking questions. In G. V. Nakamura, R. M. Taraban and D. Medin (Eds.), *The psychology of learning and motivation*, Vol. 29, Categorization by humans and machines. Orlando, FL. Academic Press, 411-436.

Graesser, A.C. and McMahen, C.L. (1993). Anomalous information triggers questions when adults solve quantitative problems and comprehend stories. *Journal of Educational Psychology*, 85, 136-151.

Graesser, A., Baggett, W., and Williams, K. (1996). Question-driven explanatory reasoning. *Applied Cognitive Psychology*, 10, 17-S32.

Greenberg, S. (2001). Context as a dynamic construct. *Human-Computer Interaction* 16(2) (December 2001), 257-268.

Greenfield, A. (2006). *Everyware: The Dawning Age of Ubiquitous Computing*. Peachpit Press, Berkeley, CA, USA.

Gregor, S. and Benbasat, I. (1999). Explanations From Intelligent Systems: Theoretical Foundations and Implications for Practice. *MIS Quarterly 23(4)*: 497–530.

Gu, T., Pung, H.K., and Zhang, D.Q. (2005). A service-oriented middleware for building context-aware services. In *Journal of Network and Computer Applications*, 28(1), 1-18.

Gupta, S., Reynolds, M.S., and Patel, S.N. (2010). ElectriSense: single-point sensing using EMI for electrical event detection and classification in the home. In *Proceedings of the 12th ACM international conference on Ubiquitous computing (Ubicomp '10)*. ACM, New York, NY, USA, 139-148.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I.H. (2009). The WEKA Data Mining Software: An Update. *SIGKDD Explorations 09*, 11(1), 10-18.

Hardian, B. (2006). Middleware support for transparency and user control in context-aware systems. In *Proceedings of the 3rd international Middleware doctoral symposium (MDS '06)*. ACM, New York, NY, USA, 4-10.

Hardian, B., Indulska, J. and Henricksen, K. (2008). Exposing Contextual Information for Balancing Software Autonomy and User Control in Context-Aware Systems. In *Proceedings of the Workshop on Context-Aware Pervasive Communities: Infrastructures, Services and Applications, affiliated with the Pervasive'2008 conference*, Sydney, May 2008.

Hastie, T. and Tibshirani, R. (1998). Classification by pairwise coupling. In *Proceedings of the 1997 conference on Advances in neural information processing systems 10 (NIPS '97)*, Michael I. Jordan, Michael J. Kearns, and Sara A. Solla (Eds.). MIT Press, Cambridge, MA, USA, 507-513.

Haynes, S.R., Cohen, M.A., and Ritter, F.E. (2009). Designs for explaining intelligent agents. *International Journal of Human-Computer Studies,* Volume 67, Issue 1, Pages 90-110.

Hempel, C.G. (1965). *Aspects of Scientific Explanation, Aspects of Scientific Explanation and Other Essays*. Free Press, New York.

Henricksen K., and Indulska, J. (2006). Developing context-aware pervasive computing applications: Models and approach. *Pervasive and Mobile Computing*, 2(1), 37-64.

Herlocker, J., Konstan, J., and Riedl, J. (2000). Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work (CSCW '00)*. ACM, New York, NY, USA, 241-250..

Hilbert, D.M. and Redmiles, D.F. (2000). Extracting usability information from user interface events. *ACM Computing Survey,* 32(4), 384-421.

Höök, K., Karlgren, J., Waern, A., Dahlbck, A., Jansson, C., Karlgren, K., and Lemaire, B. (1996). A glass box approach to adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 6(2-3), 157–184.

Höök, K. (2000). Steps to take before intelligent interfaces become real. *Interacting with Computers*, 12(4), 409–426.

Hudson, S.E. (1997). *Principles of User Interface Software: Toolkits*. Class notes. Georgia Institute of Technology, Atlanta, GA. March 1997.

Iba, W. and Langley, P. (1992). Induction of One-Level Decision Trees. In *Proceedings of the Ninth International Workshop on Machine Learning (ML '92)*, Derek H. Sleeman and Peter Edwards (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 233-240.

Jackson, M. (2000). *Problem Frames: Analyzing and Structuring Software Development Problems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Johnson, H. and Johnson, P. (1993). Explanation facilities and interactive systems. In *Proceedings of the 1st international conference on Intelligent user interfaces (IUI '93)*, Wayne D. Gray, William E. Hefley, and Dianne Murray (Eds.). ACM, New York, NY, USA, 159-166.

Kim, T. and Hinds, P.J. (2006). Who should I blame? Effects of autonomy and transparency on attributions in HRI. ROMAN 06, 80-85.

Kittur, A., Chi, E.H., and Suh, B. (2008). Crowdsourcing user studies with Mechanical Turk. In *Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems (CHI '08)*. ACM, New York, NY, USA, 453-456.

Kern, N. and Schiele, B. (2006). Towards Personalized Mobile Interruptibility Estimation. In *Proceedings of the Second international conference on Location- and Context-Awareness (LoCA'06)*, Mike Hazas, John Krumm, and Thomas Strang (Eds.). Springer-Verlag, Berlin, Heidelberg, 134-150.

Ko, A.J. and Myers, B.A. (2003). Development and evaluation of a model of programming errors. In *IEEE Symposia on Human-Centric Computing Languages and Environments 2003*, Auckland, New Zealand, 7-14.

Ko, A.J. and Myers, B.A. (2004). Designing the WhyLine: A debugging interface for asking questions about program failures. *Proceedings of the 22th international conference on Human factors in computing systems (CHI '04)*. ACM, New York, NY, USA, 151-158.

Ko, A. J. and Myers, B. A. (2009). Finding causes of program output with the Java Whyline. In *Proceedings of the 27th international conference on Human factors in computing systems (CHI '09)*. ACM, New York, NY, USA, 1569-1578.

Kofod-Petersen, A. and Aamodt, A. (2003). Case-based Situation Assessment in a Mobile Context-Aware Systems. In *Workshop on Artificial Intelligence for Mobile Systems (AIMS 2003)*.

Kofod-Petersen, A. and Mikalsen, M. (2005). Context: Representation and reasoning – Representing and reasoning about context in a mobile environment. *Revue d'Intelligence Artificielle*, 19, 479–498.

Kofod-Petersen, A., Cassens, J., and Aamodt, A. (2008). Explanatory Capabilities in the CREEK Knowledge-Intensive Case-Based Reasoner. In Proceeding of the 2008 conference on Tenth Scandinavian Conference on Artificial Intelligence: SCAI 2008, Anders Holst, Per Kreuger, and Peter Funk (Eds.). IOS Press, Amsterdam, The Netherlands, 28-35.

Kulesza T., Wong, W.K., Stumpf, S., Perona, S., White, R., Burnett, M.M., Oberst, I., and Ko. A.J. (2009). Fixing the Program My Computer Learned: Barriers for End Users, Challenges for the

Machine. In *Proceedings of the 14th international conference on Intelligent user interfaces (IUI '09).* ACM, New York, NY, USA, 187-196.

Kulesza, T., Burnett, M., Stumpf, S., Wong, W.K., Das, S., Groce, A., Shinsel, A., Bice, F., and McIntosh, K. (2011). Where are my intelligent assistant's mistakes? a systematic testing approach. In *Proceedings of the Third international conference on End-user development (IS-EUD'11)*, Maria Francesca Costabile, Yvonne Dittrich, Gerhard Fischer, and Antonio Piccinno (Eds.). Springer-Verlag, Berlin, Heidelberg, 171-186.

Kulesza, T., Stumpf, S., Burnett, M., Kwan, I. (2012). Tell Me More? The Effects of Mental Model Soundness on Personalizing an Intelligent Agent. In *Proceedings of the 30th international Conference on Human Factors in Computing Systems (CHI '12)*.. To Appear.

Lacave, C. and Díez, F.J. (2002). A review of explanation methods for Bayesian networks. *Knowledge Engineering Review* 17, 2 (June 2002), 107-127.

Lacave, C. and Díez, F.J. (2004). A review of explanation methods for heuristic expert systems. *Knowledge Engineering Review* 19, 2 (June 2004), 133-146.

Lane, N.D., Lu, H., Eisenman, S.B., and Campbell, A.T. (2009). Cooperative Techniques Supporting Sensor-Based People-Centric Inferencing. In *Proceedings of the 6th International Conference on Pervasive Computing (Pervasive '08)*, Jadwiga Indulska, Donald J. Patterson, Tom Rodden, and Max Ott (Eds.). Springer-Verlag, Berlin, Heidelberg, 75-92.

Lane, N., Lymberopoulos, D., Zhao, F., and Campbell, A.T. (2010). Hapori: context-based local search for mobile phones using community behavioral modeling and similarity. In *Proceedings of the 12th ACM international conference on Ubiquitous computing (Ubicomp '10)*. ACM, New York, NY, USA, 109-118.

Larson, E.C., Lee, T.J., Liu, S., Rosenfeld, M., and Patel, S.N. (2011). Accurate and privacy preserving cough sensing using a low-cost microphone. In *Proceedings of the 13th international conference on Ubiquitous computing (UbiComp '11)*. ACM, New York, NY, USA, 375-384.

Lee, J.D. and See, K.A. (2004). Trust in automation: designing for appropriate reliance. *Human Factors*, 42(1), 50-80.

Lee, M.L. and Dey, A.K. 2010. Embedded Assessment of Aging Adults: A Concept Validation. In *Proceedings of PervasiveHealth 2010*.

Lee, M.L. and Dey, A.K. (2011). Reflecting on pills and phone use: supporting awareness of functional abilities for older adults. In *Proceedings of the 2011 annual conference on Human factors in computing systems (CHI '11)*. ACM, New York, NY, USA, 2095-2104.

Lehnert, W.G. (1978). *The process of question-answering*. Hillsdale, NJ: Erlbaum.

Lemelson, H., King, T., and Effelsberg, W. (2008). A Study on User Acceptance of Error Visualization Techniques. In *Proceedings of the 5th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services (Mobiquitous '08)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, Belgium, Article 53, 6 pages.

Lester, J., Choudhury, T., and Borriello, G. (2006). A Practical Approach to Recognizing Physical Activities. In *Proceedings of the 4th international conference on Pervasive Computing (PERVASIVE'06)*, Kenneth P. Fishkin, Bernt Schiele, Paddy Nixon, and Aaron Quigley (Eds.). Springer-Verlag, Berlin, Heidelberg, 1-16.

Li, Y., Li, X., Ratcliffe, M., Liu, L., Qi, Y., and Liu, Q. (2011). A real-time EEG-based BCI system for attention recognition in ubiquitous environment. In *Proceedings of 2011 international workshop on Ubiquitous affective awareness and intelligent interaction (UAAII '11)*. ACM, New York, NY, USA, 33-40.

Lim, B.Y., Dey, A.K., and Avrahami, D. (2009). Why and Why Not Explanations Improve the Intelligibility of Context-Aware Intelligent Systems. In *Proceedings of the 27th international Conference on Human Factors in Computing Systems (Boston, MA, USA, April 04 - 09, 2009). CHI '09*. ACM, New York, NY, 2119-2128.

Lim, B.Y. and Dey, A.K. (2009). Assessing Demand for Intelligibility in Context-Aware Applications. In *Proceedings of the 11th international Conference on Ubiquitous Computing (Orlando, Florida, USA, September 30 - October 03, 2009). Ubicomp '09*. ACM, New York, NY, 195-204.

Lim, B.Y. and Dey, A.K. (2010). Toolkit to Support Intelligibility in Context-Aware Applications. In *Proceedings of the 12th ACM international Conference on Ubiquitous Computing (Copenhagen, Denmark, September 26 - 29, 2010). Ubicomp '10*. ACM, New York, NY, 13-22.

Lim, B.Y., Brdiczka, O., and Bellotti, V. (2010). Show me a good time: using content to provide activity awareness to collaborators with activityspotter. In *Proceedings of the 16th ACM international conference on Supporting group work (GROUP '10)*. ACM, New York, NY, USA, 263-272.

Lim, B.Y. and Dey, A.K. (2011). Design of an Intelligible Mobile Context-Aware Application. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI '11)*. ACM, New York, NY, USA, 157-166.

Lim, B.Y. and Dey, A.K. (2011). Investigating Intelligibility for Uncertain Context-Aware Applications. In *Proceedings of the 13th international conference on Ubiquitous computing (UbiComp '11)*. ACM, New York, NY, USA, 415-424.

Lim, B. Y., Dey, A. K. (2012). Field Evaluation of an Intelligible Context-Aware Application. *CMU-HCII Technical Report*.

Lu, H., Pan, W., Lane, N.D., Choudhury, T., and Campbell, A.T.. (2009). SoundSense: scalable sound sensing for people-centric applications on mobile phones. In *Proceedings of the 7th international conference on Mobile systems, applications, and services (MobiSys '09)*. ACM, New York, NY, USA, 165-178.

Ma, T., Kim, Y.D., Ma, Q., Tang, M., Zhou, W. (2005). Context-aware implementation based on cbr for smart home. In *Proceedings of Wireless And Mobile Computing, Networking And Communications (WiMob 2005)*. IEEE, IEEE Computer Society, 112–115.

Madan, A., Cebrian, M., Lazer, D., and Pentland, A. (2010). Social sensing for epidemiological behavior change. In *Proceedings of the 12th ACM international conference on Ubiquitous computing (Ubicomp '10)*. ACM, New York, NY, USA, 291-300.

Madigan, D., Mosurski, K., and Almond, R. (1996). Explanation in Belief Networks. In *Journal of Computational and Graphical Statistics*, 6(2), pp. 160-181.

Mankoff, J., Dey, A.K., Hsieh, G., Kientz, J., Lederer, S., and Ames, M. (2003). Heuristic evaluation of ambient displays. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '03)*. ACM, New York, NY, USA, 169-176.

McCreath, E., Kay, J., Crawford, E. (2006). IEMS-an approach that combines hand crafted rules with learnt instance based rules. *Australian Journal of Intelligent Information Processing Systems,* 9(1), 49-63.

McGuinness, D. L., Glass, A., Wolverton, M., and Pinheiro da Silva, P. (2007). A Categorization of Explanation Questions for Task Processing Systems. In *AAAI Workshop on Explanation-Aware Computing (ExaCt-07)*.

McSherry, D. (2005). Explanation in Recommender Systems. *Artificial Intelligence*. 24(2), 179-197.

Metaxas, G., Metin, B., Schneider, J., Markopoulos, P., De Ruyter, B. (2007). Daily Activities Diarist: Supporting Aging in Place with Semantically Enriched Narratives. In *Proceedings of INTERACT 2007*, Heidelberg, Germany, Springer Verlag, 390-403.

Metaxas, G. (2010). End-User Programming of Awareness Systems. *Ph.D. Thesis*. December 2010. User System Interaction Programme, Einhoven University of Technology.

Milewski, A.E. and Smith, T.M. (2000). Providing Presence Cues to Telephone Users. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work (CSCW '00)*. ACM, New York, NY, USA, 89-96.

Miluzzo, E., Lane, N.D., Fodor, K., Peterson, R., Lu, H., Musolesi, M., Eisenman, S.H., Zheng, X., and Campbell, A.T. (2008). Sensing meets mobile social networks: the design, implementation and evaluation of the CenceMe application. In *Proceedings of the 6th ACM conference on Embedded network sensor systems (SenSys '08)*. ACM, New York, NY, USA, 337-350.

Motorola Smart Actions. http://www.motorola.com/blog/2011/11/17/motorola-smart-actions-uncover-the-secret-super-powers-of-your-phone/. Retrieved 17 November 2011.

Mozina M., Demsar, J., Kattan, M., and Zupan, B. (2004). Nomograms for Visualization of Naive Bayesian Classifier. In *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD '04)*. Springer-Verlag New York, Inc., New York, NY, USA, 337-348.

Muir, B. (1994). Trust in automation: Part i. theoretical issues in the study of trust and human intervention in automated systems. *Ergonomics*, 37(11): 1905–1922.

Muir, B.M. and Moray, N. (1996). Trust in automation. Part II. Experimental studies of trust and human intervention in a process control simulation. *Ergonomics*, 39(3), 429-460.

Murao, K., Terada, T., Takegawa, Y., and Nishio, S. (2009). A Context-Aware System that Changes Sensor Combinations Considering Energy Consumption. In *Proceedings of the 6th International Conference on Pervasive Computing (Pervasive '08)*, Jadwiga Indulska, Donald J. Patterson, Tom Rodden, and Max Ott (Eds.). Springer-Verlag, Berlin, Heidelberg, 197-212.

Myers, B.A., Weitzman, D., Ko, A.J., and Chau, D.H. (2006). Answering Why and Why Not Questions in User Interfaces. In *Proceedings of the 24th international conference on Human factors in computing systems (CHI '06)*. ACM, New York, NY, USA, 397-406.

Mynatt, E. D., Rowan, J., Craighill, S., and Jacobs, A. (2001). Digital family portraits: supporting peace of mind for extended family members. In *Proceedings of the 19th international conference on Human factors in computing systems (CHI '01)*. ACM, New York, NY, USA, 333-340.

Newcomb, E., Pashley, T., and Stasko, J. (2003). Mobile computing in the retail arena. Mobile computing in the retail arena. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '03)*. ACM, New York, NY, USA, 337-344.

Newman, M.W. Sedivy, J.Z., Neuwirth, C.M., Edwards, W.K., Hong, J.I., Izadi, S., Marcelo, K., and Smith, T.F. (2002). Designing for serendipity: supporting end-user configuration of ubiquitous computing environments. In *Proceedings of the 4th conference on Designing interactive systems: processes, practices, methods, and techniques (DIS '02)*. ACM, New York, NY, USA, 147-156.

Ngair, T.H. (1993). A new algorithm for incremental prime implicate generation. In *Proceedings of the 13th international joint conference on Artificial intelligence (IJCAI'93)*, 1. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 46-51.

Niculescu-Mizil, A., and Caruana, R. (2005). Obtaining calibrated probabilities from boosting. In *Proceedings of the 21st conference on uncertainty in artificial intelligence*. Corvallis: AUAI Press.

Norman, Donald A. (1988). *The Design of Everyday Things*. New York, Doubleday.

Nugent, C. and Cunningham, P. (2005). A case-based explanation system for black-box systems. *Artificial Intelligence Review*, 24(2), 163–178.

Núñez, H., Angulo, C., and Català, A. (2002). Rule extraction from support vector machines. In *Proceedings of European Symposium on Artificial Neural Networks*, 107-112.

Oliver, E.A. and Keshav, S. (2011). An empirical approach to smartphone energy level prediction. In *Proceedings of the 13th international conference on Ubiquitous computing (UbiComp '11*). ACM, New York, NY, USA, 345-354.

Olsen, D. R. (2007). Evaluating user interface systems research. In *Proceedings of the 20th annual ACM symposium on User interface software and technology (UIST '07)*. ACM, New York, NY, USA, 251-258.

Orlikowski, W. J. (2000). Using Technology and Constituting Structures: A Practice Lens for Studying Technology in Organizations. *Organization Science* 11(4), 404-428.

Oulasvirta, A. (2005). Grounding the Innovation of Future Technologies. *Human Technology*, 1 (1):58-75.

Parasuraman, R. and Miller, C.A. (2004). Trust and etiquette in high-criticality automated systems. *Communications ACM*, 47, 4 (April 2004), 51-55.

Patel, K, Fogarty, J., Landay, J.A., and Harrison, B. (2008). Investigating statistical machine learning as a tool for software development. In *Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems (CHI '08)*. ACM, New York, NY, USA, 667-676.

Platt, J.C. (1998). Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. In *Advances in Kernel MethodsSupport Vector Learning*. *Technical Report MSR-TR-98-14*, Microsoft Research, 1998.

Platt, J.C. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, A. J. Smola, P. Bartlett, B. Schölkopf and D. Schuurmans (eds.), 61–74. Cambridge, MIT Press, 1999.

Price, Jim. (2007). Understanding dB. *Professional Audio*. Retrieved 27 April 2012.

Poulin, B., Eisner, R., Szafron, D., Lu, P., Greiner, R., Wishart, D. S., Fyshe, A., Pearcy, B., MacDonnell, C., and Anvik, J. (2006). Visual explanation of evidence in additive classifiers. In *Proceedings of the 18th conference on Innovative applications of artificial intelligence - Volume 2 (IAAI'06)*, Bruce Porter (Ed.), Vol. 2. AAAI Press 1822-1829.

Pu, P. and Chen, L. (2006). Trust building with explanation interfaces. In *Proceedings of the 11th international conference on Intelligent user interfaces (IUI '06)*. ACM, New York, NY, USA, 93-100.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers.

Rabiner L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, 77(2):257–286, 1989.

Rimassa, G., Greenwood, D., and Calisti, M. (2005). Palpable computing and the role of agent technology. In *Proceedings of the 4th international Central and Eastern European conference on Multi-Agent Systems and Applications (CEEMAS'05)*, Michael Pěchouček, Paolo Petta, and László Zsolt Varga (Eds.). Springer-Verlag, Berlin, Heidelberg, 1-10.

Robnik-Šikonja, M. and Kononenko, I. (2008). Explaining Classifications For Individual Instances. In *IEEE Transactions on Knowledge and Data Engineering*, 20(5): 589-600.

Rosenthal, S., Dey, A.K., and Veloso, M. (2011). Using Decision-Theoretic Experience Sampling to Build Personalized Mobile Phone Interruption Models. In *Proceedings of the 9th international conference on Pervasive computing (Pervasive'11)*, Kent Lyons, Jeffrey Hightower, and Elaine M. Huang (Eds.). Springer-Verlag, Berlin, Heidelberg, 170-187.

Roth-Berghofer, T.R. (2004) Explanations and case-based reasoning: foundational issues. In *Proceedings of Advances in Case-Based Reasoning*. Springer-Verlag, Berlin, Heidelberg, Paris, 389-403.

Roto, V., Oulasvirta, A., Haikarainen, T., Kuorelahti, J., Lehmuskallio, H., and Nyyssönen, T. (2004). Examining Mobile Phone Use in the Wild with Quasi-Experimentation. *Helsinky Institute for Information Technology (HIIT), Technical Report*, 1, 2004.

Rukzio, E., Hamard, J., Noda, C., and Luca, A.D. (2006). Visualization of uncertainty in context aware mobile applications. In *Proceedings of the 8th conference on Human-computer interaction with mobile devices and services (MobileHCI '06)*. ACM, New York, NY, USA, 247-250.

Schapire, R.E. and Singer, Y. 1999. Improved Boosting Algorithms Using Confidence-rated Predictions. *Machine Learning*, 37, 3 (December 1999), 297-336.

Schilit, B.N., Adams, N.I., and Want, R. (1994) Context-aware computing applications. In *Proceedings of the 1st International Workshop on Mobile Computing Systems and Applications*, pp. 85–90.

Shinsel, A., Kulesza, T., Burnett, M., Curran, W., Groce, A., Stumpf, S., and Wong, W.K. (2011). Mini-Crowdsourcing End-User Assessment of Intelligent Assistants: A Cost-Benefit Study. In *Proceedings of the 2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 47-54.

Schonlau, M., Soest, A. V., Kapteyn, A., and Couper, M. P. (2006). Selection bias in web surveys and the use of propensity scores. Sociological Methods & Research, 37(3): 291-318.

Silveira, M.S., de Souza, C.S., and Barbosa, S.D.J. (2001). Semiotic engineering contributions for designing online help systems. In *Proceedings of the 19th annual international conference on Computer documentation (SIGDOC '01)*. ACM, New York, NY, USA, 31-38.

Shortliffe, E. H. (1975). Mycin: a Rule-Based Computer Program for Advising Physicians Regarding Antimicrobial Therapy Selection. *Doctoral Thesis*. UMI Order Number: AAI7513600., Stanford University.

Sohn, T., Varshavsky, A., LaMarca, A., Chen, M.Y., Choudhury, T., Smith, I., Consolvo, S., Hightower, J., Griswold, W.G., and de Lara, E. (2006). Mobility detection using everyday GSM traces. In *Proceedings of the 8th international conference on Ubiquitous Computing (UbiComp'06)*, Paul Dourish and Adrian Friday (Eds.). Springer-Verlag, Berlin, Heidelberg, 212-224.

Sørmo, F., Cassens, J., and Aamodt, A. (2005). Explanation in Case-Based Reasoning — Perspectives and Goals. *Artificial Intelligence*, 24(2), 109-143.

Spieker, P. (1991). Natürlichsprachliche Erklärungen in technischen Expertensystemen. *Dissertation*, University of Kaiserslautern.

Strauss, A.L. and Corbin, J.M. (1990). *Basics of Qualitative Research: Grounded theory procedures and techniques*. Newbury Park, CA: Sage Publications.

Stumpf, S., Rajaram, V., Li, L., Wong, W.K., Burnett, M., Dietterich, T., Sullivan, E., and Herlocker, J. (2009). Interacting meaningfully with machine learning systems: Three experiments. *International Journal of Human-Computer Studies*. 67, 8 (August 2009), 639-662.

Swartout, W. R. (1983). What Kind of Expert Should a System Be? XPLAIN: A System for Creating and Explaining Expert Consulting Programs. *Artificial Intelligence*, (21), 285-325.

Swartout, W. R., and Smoliar, S. W. (1987). On Making Expert Systems More Like Experts. *Expert Systems*,4(3), 196-207.

Sun, L., Zhang, D., and Li, N. (2011). Physical activity monitoring with mobile phones. In *Proceedings of the 9th International Conference On Smart homes and health Telematics (ICOST'11)*, Bessam Abdulrazak, Sylvain Giroux, Hélène Pigot, Bruno Bouchard, and Mounir Mokhtari (Eds.). Springer-Verlag, Berlin, Heidelberg, 104-111.

Tapia, E. M., Intille, S. S., and Larson, K. (2004). Activity Recognition in the Home Using Simple and Ubiquitous Sensors. In *Proceedings of the 2nd international conference on Pervasive computing (Pervasive'04)*, Alois Ferscha and Friedemann Mattern (Eds.). Springer-Verlag, Berlin, Heidelberg, 158-175.

Terada, T., Tsukamoto, M., Hayakawa, K., Yoshihisa, T., Kishino, Y., Kashitani A., and Nishio, S. (2004). Ubiquitous Chip: A Rule-Based I/O Control Device for Ubiquitous Computing. In *Proceedings of the 2nd international conference on Pervasive computing (Pervasive'04)*, Alois Ferscha and Friedemann Mattern (Eds.). Springer-Verlag, Berlin, Heidelberg, 238-253.

Tickle, A.B., Andrews, R., Golea, M., and Diederich, J. (1998). The truth will come to light: Directions and challenges in extracting the knowledge embedded within trained artificial neural networks. *IEEE Transactions on Neural Networks*, 9(6), 1057-1068.

Tintarev, N. (2007). Explaining Recommendations. In *Proceedings of the 11th international conference on User Modeling* (UM '07), Cristina Conati, Kathleen Mccoy, and Georgios Paliouras (Eds.). Springer-Verlag, Berlin, Heidelberg, 470-474.

Tseitin, G.S. (1968). On the complexity of derivation in propositional calculus. In Studies in Constructive Mathematics and Mathematical Logic, Part II, *A. Slisenko, Ed. Seminars in Mathematics, V.A. Steklov, Mathematical Institute, Leningrad, vol. 8. Consultants Bureau*, 115-125. English translation appears in *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970*, J. Siekmann and G. Wrightson, Eds. Springer (1983), 466-483.

Tsukada, K. and Yasumura, M. (2004). ActiveBelt: Belt-Type Wearable Tactile Display for Directional Navigation. In *Proceedings of the 6th international conference on Ubiquitous computing (UbiComp '04)*, 384-399.

Tullio, J., Dey, A.K., Fogarty, J., and Chalecki, J. (2007). How it works: A field study of non-technical users interacting with an intelligent system. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '07)*. ACM, New York, NY, USA, 31-40.

Van der Meik, H. (1987). Assumptions of information-seeking questions. *Questioning Exchange*, 1, 111-118.

van Kasteren, T., Noulas, A., Englebienne, G., and Kröse, B. (2008). Accurate Activity Recognition in a Home Setting. In *Proceedings of the 10th international conference on Ubiquitous computing (UbiComp '08)*. ACM, New York, NY, USA, 1-9.

Van Melle, W., Shortliffe, E.H., and Buchanan, B.G. (1984). Emycin: A knowledge-engineer's tool for constructing rule-based expert systems. In E.H. Shortliffe and B.G. Buchanan, editors, *Rule-Based Expert Systems*, Addison- Wesley, 302-313.

Vermeulen, J., Slenders, J., Luyten, K., and Coninx, K. (2009). I Bet You Look Good on the Wall: Making the Invisible Computer Visible. In *Proceedings of the European Conference on Ambient Intelligence (AmI '09)*, Manfred Tscheligi, Boris Ruyter, Panos Markopoulus, Reiner Wichert, Thomas Mirlacher, Alexander Meschterjakov, and Wolfgang Reitberger (Eds.). Springer-Verlag, Berlin, Heidelberg, 196-205.

Vermeulen, J. (2010). Improving intelligibility and control in Ubicomp. In *Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing (Ubicomp '10)*. ACM, New York, NY, USA, 485-488.

Vermeulen, J., Vanderhulst, G., Luyten, K., and Coninx, K. (2010). PervasiveCrystal: Asking and Answering Why and Why Not Questions about Pervasive Computing Applications. In *Proceedings of the 2010 Sixth International Conference on Intelligent Environments (IE '10)*. IEEE Computer Society, Washington, DC, USA, 271-276.

Vurgun, S., Philipose, M., and Pave, M. (2007). A statistical reasoning system for medication prompting. In *Proceedings of the 9th international conference on Ubiquitous computing (UbiComp '07)*, John Krumm, Gregory D. Abowd, Aruna Seneviratne, and Thomas Strang (Eds.). Springer-Verlag, Berlin, Heidelberg, 1-18.

Ware, C. (2000). *Information Visualization: Perception for design*. San Francisco, CA: Morgan Kaufmann.

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, 265(3): 94-104.

Weiser, M. and Brown, J.S. (1997). The coming age of calm technology. *Beyond Calculation: the Next Fifty Years*, 75-85.

Weka 3: Data Mining Software in Java. http://www.cs.waikato.ac.nz/ml/weka/. Retrieved 27th January 2012.

Weka for Android. https://github.com/rjmarsan/Weka-for-Android. Retrieved 26th August 2011.

Welbourne, E., Balazinska, M., Borriello, G., and Fogarty, J. (2010). Specification and Verification of Complex Location Events. In *Proceedings of the 8th international conference on Pervasive computing (Pervasive'10)*, 57-75.

Wick, M.R., and Slagle, J.R. (1989). An explanation facility for today's expert systems. *IEEE Expert: Intelligent Systems and Their Applications*, 4(1), 26-36.

Wilson, D.H. and Atkeson, C. (2005). Simultaneous tracking and activity recognition (STAR) using many anonymous, binary sensors. In *Proceedings of the Third international conference on Pervasive Computing (Pervasive'05)*, Hans-W. Gellersen, Roy Want, and Albrecht Schmidt (Eds.). Springer-Verlag, Berlin, Heidelberg, 62-79.

Wong, W.K., Oberst, I., Das, S., Moore, T., Stumpf, S., McIntosh, K., and Burnett, M. (2011). End-user feature labeling: a locally-weighted regression approach. In *Proceedings of the 16th* *international conference on Intelligent user interfaces (IUI '11)*. ACM, New York, NY, USA, 115-124.

Yan, Z., Liu, C., Niemi, V., and Yu, G. (2010). Effects of displaying trust information on mobile application usage. In *Proceedings of the 7th international conference on Autonomic and trusted computing (ATC'10)*, Bing Xie, Juergen Branke, S. Masoud Sadjadi, Daqing Zhang, and Xingshe Zhou (Eds.). Springer-Verlag, Berlin, Heidelberg, 107-121.

Zheng, Y., Li, Q., Chen, Y., Xie, X., and Ma, W.Y. (2008). Understanding mobility based on GPS data. In *Proceedings of the 10th international conference on Ubiquitous computing (UbiComp '08)*. ACM, New York, NY, USA, 312-321.

Zheng, Y., Chen, Y., Li, Q., Xie, X., and Ma, W.Y. (2010). Understanding transportation modes based on GPS data for web applications. *ACM Transactions on the Web*, 4(1), 1-36.

Zimmermann, A. (2003). Context-awareness in user modelling: requirements analysis for a case-based reasoning application. In *Proceedings of the 5th international conference on Case-based reasoning: Research and Development (ICCBR'03)*, Kevin D. Ashley and Derek G. Bridge (Eds.). Springer-Verlag, Berlin, Heidelberg, 718-732.

A Intelligibility Toolkit Software

The Intelligibility Toolkit is released as open source under the GNU General Public License (GPL)[5].

A website for the updated Context Toolkit and the Intelligibility Toolkit is located at:

http://contexttoolkit.org[6]

It contains:

- A description of the Context Toolkit and Intelligibility research
- A list of publications related to Context Toolkit and Intelligibility in Context-Aware Applications
- Documentation for the software, including
  - Description of the Context Toolkit and Intelligibility Toolkit components
  - Download and Installation
  - Javadoc API Documentation
  - Tutorials on how to use both toolkits
  - Demonstration Applications
- Information on how to download the source code and binaries

[5] GNU General Public License. http://www.gnu.org/licenses/gpl.html. Retrieved 4 April 2012.

[6] Retrieved 4 April 2012.

[7] Circular Error Probable. http://en.wikipedia.org/wiki/Circular_error_probable. Retrieved 27 April 2012.

B Intelligibility Toolkit Explainers

We provide the theoretical definitions, mathematical proofs detailing the derivations and explanation generation algorithms of the explainers implemented in the Intelligibility Toolkit. The explainers explain inference, not how the models were learned.

B.1 Abstract Base Explainer

To align with later terminology in this appendix, we use the term *class* to refer to a discrete (or nominal) output value. We use the term *feature* to refer to an input. We shall also use the term *infer* to refer to the concept of the context-aware model reasoning or deciding. These terms are commonly used in the machine learning literature.

B.1.1 What Explanation

We describe the event that the ith class was inferred as

$$Y_i: \quad y = y_i$$

where y is the output variable, and y_i is the output value of the ith class.

The What explanation returns y_i.

B.1.2 Inputs Explanation

We denote the inputs to the inference model as a vector of n feature values:

$$x = \bigcap_{r=1}^{n} x_r = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \tag{B.1}$$

where x_r is value of the rth input feature.

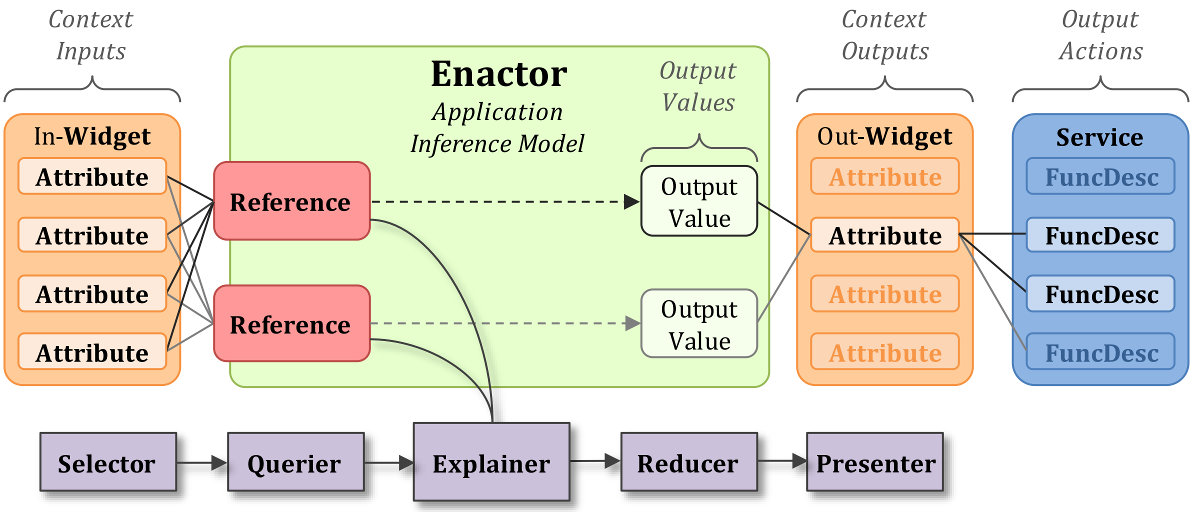
B.1.3 OUTPUTS EXPLANATION

We denote the output values from the inference model as a vector of m possible values:

$$y = \bigcup_{j=0}^{m} y_j \tag{B.2}$$

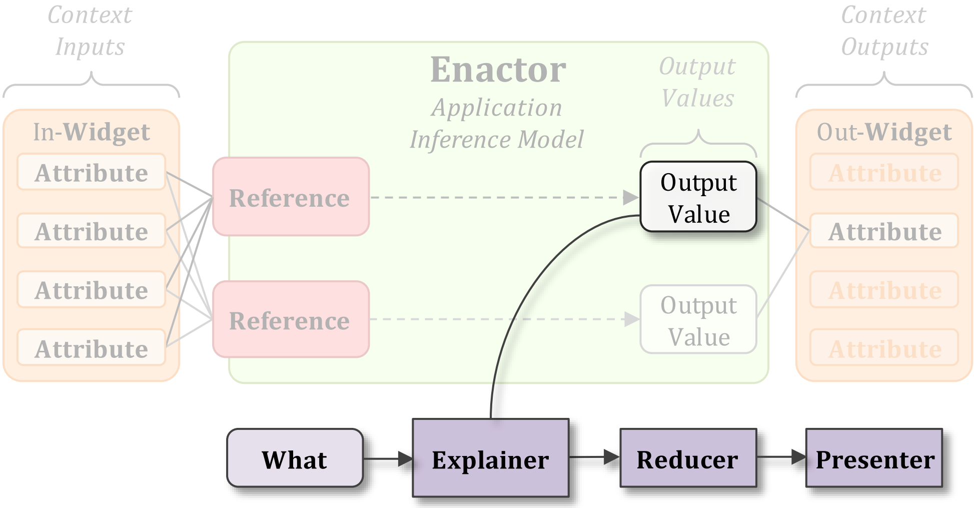
where y_j is the jth class value.

B.2 ENACTOR BASE EXPLAINER



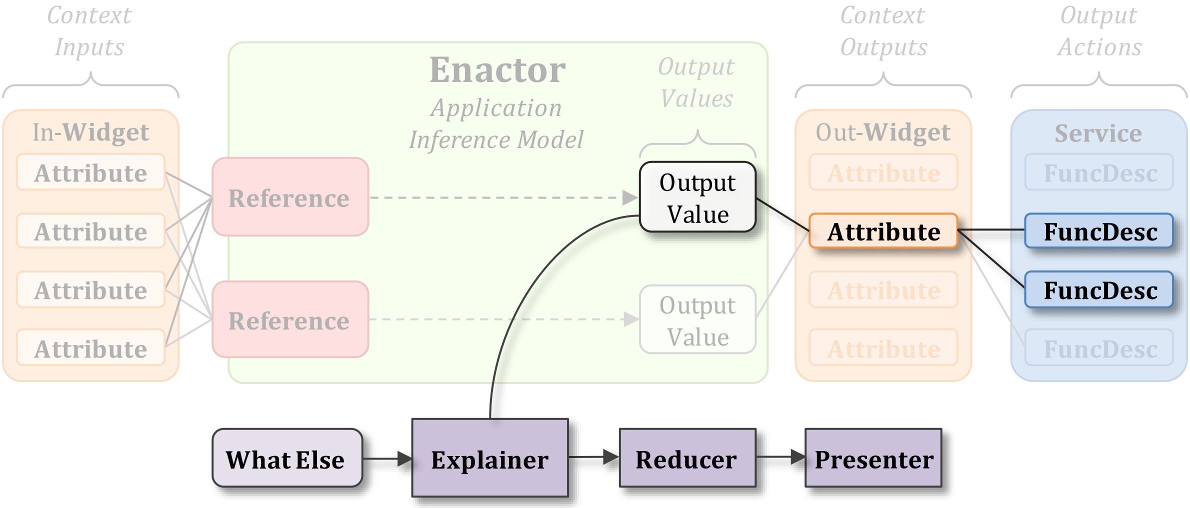
The base Explainer generates explanation types that are model-independent. Particularly, these explanations depend on the underlying architecture or toolkit for building the context-aware application, but they do not depend on the inference model. Here, we describe the algorithms for generating the explanation types from the Context Toolkit.

B.2.1 WHAT EXPLANATION



The What explanation retrieves the triggered output value in the Enactor.

B.2.2 WHAT ELSE EXPLANATION

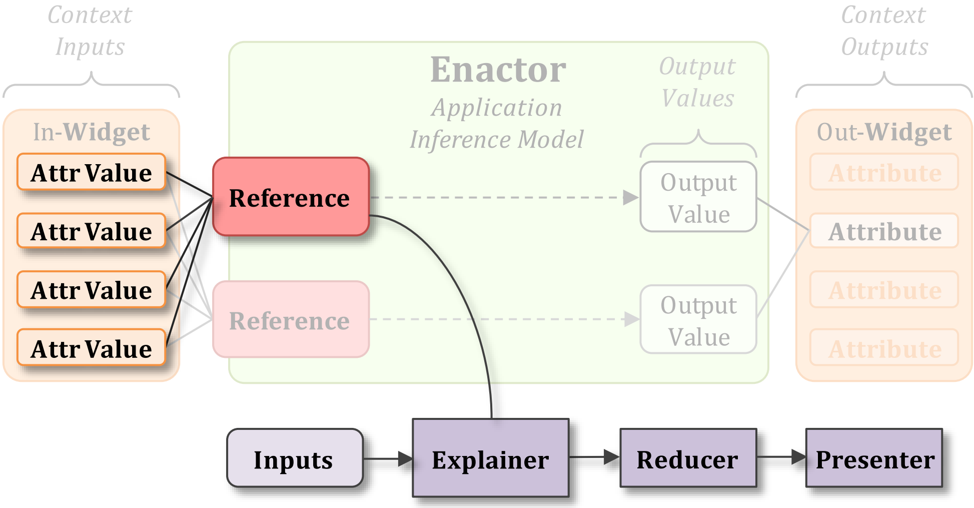


The What Else explanation traces the `Attribute` in the `Out-Widget` that is changed, and the associated `FunctionDescriptions` corresponding to the actions or services that were performed.

B.2.3 WHEN EXPLANATION

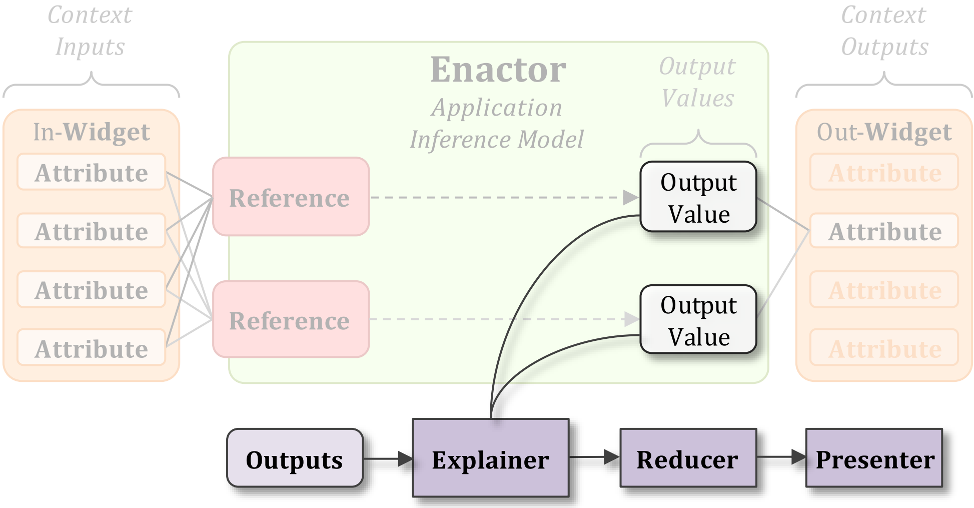
The When explanation retrieves the timestamp of when the value was updated.

B.2.4 INPUTS EXPLANATION



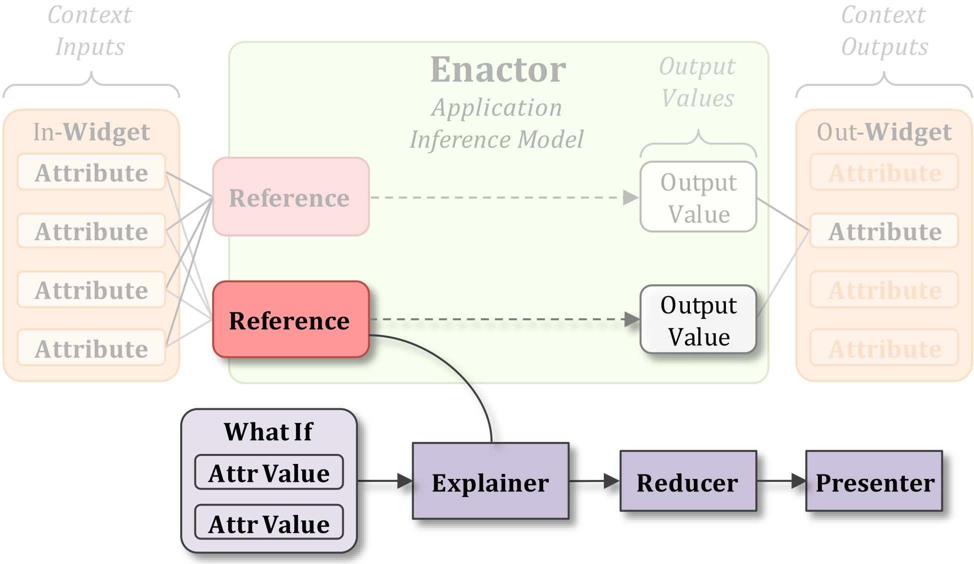
The Inputs explanation selects the `EnactorReference` that was triggered, and the corresponding `WidgetState` of `Attribute` values from the In-`Widget`.

B.2.5 OUTPUTS EXPLANATION



The Outputs explanation lists all output values that may be generated from the `Enactor`.

B.2.6 WHAT IF EXPLANATION



The What If explanation takes in a `InputQuery` with user specified input `Attribute` values, combines them with the actual sensed attribute values that were unspecified, and triggers the corresponding `EnactorReference` and output value.

B.3 WEKA BASE EXPLAINER

We implement some algorithms for model-independent explanations to support the use of the Intelligibility Toolkit for developers who use the Weka framework, but not the Context Toolkit. It supports explanations for What, What If, Inputs, Outputs, and Certainty, but does not implement explanations for When, and History.

In fact, we have decoupled the Intelligibility Toolkit from the Context Toolkit to make it more portable, *e.g.*, such as porting to Android phones.

B.3.1 WHAT AND WHAT IF EXPLANATIONS

The What explanation is essentially the inference made on the input `Instance`. This is obtained by calling the `Classifier.classifyInstance(Instance)` and `Instance.value(double)`. On its own a classifier model does not perform any action or service, so the What explanation is equivalent to the What If explanation.

B.3.2 INPUTS EXPLANATION

The Inputs explanation is obtained from taking the input `Instance` that was passed to the classifier, and converting it to an Expression `DNF` form with one `Reason` consisting of input features as each element.

B.3.3 OUTPUTS EXPLANATION

In the Weka framework, each Instance contains information about its class Attribute. This Attribute, in turn, contains information about the possible values it can take. By calling `Attribute.values()`, we can obtain the Outputs information that we present in the Outputs explanation.

B.3.4 CERTAINTY EXPLANATION

For each classification a Weka classifier makes of an instance, it can provide the probability distribution for all class values. This distribution can be retrieved by calling the `distributionForInstance(Instance)` method, and it returns the probability of or certainty for inferring the outcome of the input instance as each of the class values.

B.3.5 MODEL-BASED EXPLANATIONS

The remaining explanation types depend on the individual algorithms of each classifier. We have implemented Weights of Evidence explainers for all currently-supported classifiers in the Intelligibility Toolkit. Furthermore, the Decision Tree classifier, J48, has a reasoning trace explainer.

B.4 RULE TRACE EXPLAINER

This explainer seeks to provide an explanation by tracing the inference path through a set of rules and considering paths that were or were not triggered.

We consider rules consisting of condition literals (*e.g.*, temperature < 10, volume ≥ 55, $x = 25$x) which may be combined with Boolean operators AND (represented with '$\cap$' or as a product '$\cdot$', '$\times$' in our notation) and OR (represented with '$\cup$' or as a sum '$+$'). Hence, each rule or multiple rules can be considered a Boolean expression, ϕ. To be able to generate explanations from ϕ, we choose to convert them into disjunctive normal form (DNF), ψ, which is very suitable for automated deduction:

We define an explanation in terms of one or multiple reasons (*e.g.*, multiple reasons for Why Not). Each reason can be a singular conditional (*e.g.*, one certainty value for a Certainty explanation) or a conjunction (*e.g.*, multiple conditionals for a Why explanation). The conditional is the atomic unit of an explanation (*e.g.*, certainty=90%, temperature<24°C). Furthermore, there can be negated conditionals (*e.g.*, not temperature≥24°C). Formally, we define explanations in Disjunctive Normal Form (DNF), *i.e.* a *disjunction* (OR) of *conjunctions* (ANDs) of condition *literals* (see example in Figure 6.7). The standardization of explanation information supports R4 such that there is a standard way to pipe and feed different explanation types.

$$\phi \xrightarrow{\text{to DNF}} \psi$$

$$\psi = \mathrm{ToDNF}(\phi) = \bigcup_v \xi_v = \bigcup_v \bigcap_\rho X_{v\rho} \tag{B.3}$$

where

$$\psi = \bigcup_v \xi_v \text{ is the DNF tree form of the rules,} \tag{B.4}$$

$$\xi_v = \bigcap_{\rho_v = 1} X_{v\rho} \text{ is the } v\text{th rule trace } \textit{clause} \text{ in that DNF, and} \tag{B.5}$$

$$X_{v\rho} \text{ is the } \rho_v\text{th condition } \textit{literal} \text{ in that rule trace.}$$

Figure B.1 illustrates a diagrammatic representation of ψ.

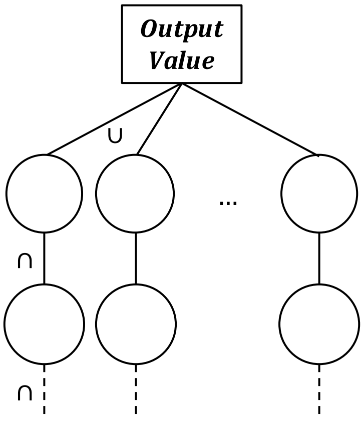


Figure B.1: Diagram representation of the Explanation Data Structure in Disjunctive Normal Form (DNF). Each circular node represents a condition literal, each vertical column represents a conjunction of literals, and these are joined horizontally as a disjunction.

We cover a naïve algorithm to perform such a conversion in Section B.4.1.

B.4.1 NAÏVE CONVERSION OF ARBITRARY BOOLEAN EXPRESSION TO DNF

A common strategy to convert an arbitrary Boolean expression to disjunctive normal form (DNF) is to first convert it to negation normal form (NNF), φ:

$$\phi \xrightarrow{\text{to NNF}} \varphi \xrightarrow{\text{to DNF}} \psi$$
$$\psi = \text{NNFtoDNF}\big(\text{ToNNF}(\phi)\big) \tag{B.6}$$

First, we convert the arbitrary Boolean expression, ϕ, to Negation Normal Form (NNF) by "pushing negations in" to the literal terms, then applying the following logical equivalences

De Morgan's laws

$$\overline{A+B} \equiv \bar{A} \cdot \bar{B} \quad , \quad \overline{AB} \equiv \bar{A} + \bar{B}$$
$$\overline{\sum\nolimits_v A_v} \equiv \prod\nolimits_v \overline{A_v} \quad , \quad \overline{\prod\nolimits_v A_v} \equiv \sum\nolimits_v \overline{A_v} \tag{B.7}$$

Double negative elimination

$$\bar{\bar{A}} \equiv A \tag{B.8}$$

Next, we convert the NNF to DNF by applying the logical equivalence:

Distributivity

$$A + BC \equiv (A+B)(A+C)$$
$$A(B+C) \equiv AB + AC \tag{B.9}$$

To convert to DNF, we only need the first rule in Equation (B.9). Figure B.2 and Figure B.3 describe the algorithms for converting an arbitrary Boolean expression to DNF.

```
$\phi$ = original Boolean expression
$\varphi$ = ToNNF($\phi$) // Boolean expression in NNF

ToNNF($\phi$)
    If $\phi = \bar{A}$ for some $A$ // negation
        If $A = \bar{B}$ for some $B$ // double negation, i.e., $\phi = \bar{\bar{B}} = B$
            Return ToNNF($B$) // double negation elimination
        If $A = \sum_v B_v$ for some $B_v$ // disjunction
            Return $\prod_\rho$ ToNNF($\overline{B_v}$) // apply De Morgan's law: $\overline{\sum_v B_v} = \prod_v \overline{B_v}$
        If $A = \prod_v B_v$ for some $B_v$ // conjunction
            Return $\sum_v$ ToNNF($\overline{B_v}$) // apply De Morgan's law: $\overline{\prod_v B_v} = \sum_v \overline{B_v}$
    Else
        If $A = \sum_v B_v$ for some $B_v$ // disjunction
            Return $\sum_v$ ToNNF($B_v$) // recursively apply ToNNF
        If $A = \prod_v B_v$ for some $B_v$ // conjunction
            Return $\prod_v$ ToNNF($B_v$) // recursively apply ToNNF

    // terminal literal
    Return $\phi$ // return self
```

Figure B.2. Recursive algorithm to convert an arbitrary Boolean expression to NNF.

```
$\varphi$ = Boolean expression in NNF
$\xi$ := new empty trace clause
$\psi$ := new empty DNF // gets updated in NNFtoDNF
NNFtoDNF($\varphi, \xi, \psi$)

NNFtoDNF($\varphi, \xi, \psi$)
    If $\varphi = \sum_v B_v$ for some $B_v$ // disjunction
        For each $B_v$
            $\xi_v$ := Clone $\xi$ // duplicate trace to branch for each disjunction path
            NNFtoDNF($B_v, \xi_v, \psi$) // recurse for each disjunction branch
    Else If $\varphi = \prod_v B_v$ for some $B_v$ // conjunction
        For each $B_v$
            NNFtoDNF($B_v, \xi, \psi$) // recurse to extend this path
    Else // $\varphi$ is terminal literal $\Longrightarrow$ reached leaf
        $X := \varphi$ // convert terminal Boolean expression to condition literal
        $\xi := \xi + X$ // append literal $X$ to trace clause $\xi$
        $\psi := \psi + \xi$ // append trace clause $\xi$ to DNF $\psi$
```

Figure B.3. Recursive algorithm to convert a Boolean expression in NNF to DNF.

In general, converting an arbitrary Boolean expression into DNF can be expensive (*e.g.*, converting CNF to DNF can cause an exponential blow-up in size [Ngair, 1993]). However, we assume that these rules will be handcrafted by end-users or designers of applications, so they should not be overly complicated. For example, Motorola's Smart Actions [Motorola, 2011] only allows end-users to write individual rules with a set of triggers that may be activated, *i.e.*, each rule is a conjunction of condition literals.

B.4.2 LOGIC MINIMIZATION TO SIMPLIFY OF RULES

The conversion to DNF does not guarantee that the DNF tree, ψ, will be simple or minimized. It may contain much redundancy and repetition of literals or clauses. We can employ `QmcReducer` (see Section 6.11.1.3), which uses the Quine-McCluskey algorithm to minimize Boolean functions

$$\psi_i \leftarrow \text{QmcMinimize}(\psi_i) \tag{B.10}$$

To save on future computation when explanations are asked, this minimization may be performed when parsing the rules into DNF. However, this compromises the original structure of the rules, which end-users may have hand-crafted, and therefore may be more intelligible than in a compressed, minimized form.

B.5 DISJUNCTIVE NORMAL FORM (DNF) EXPLAINER

We keep separate the rules that infer different classes, *i.e.*, $\phi_i \cap \phi_j = \emptyset$, for $i \neq j$. For an inference model with m possible class values, we will store a DNF rule tree for each class value, j, in a *map* of DNF trees:

$$\Psi = \bigcup_{j=1}^{m} \psi_j \tag{B.11}$$

where

$$\psi_j = \bigcup_{v_j} \xi_{jv} \text{ is the DNF tree for all rules that infer the } j\text{th class,} \tag{B.12}$$

$$\xi_{jv} = \bigcap_{\rho_{jv}} X_{jv\rho} \text{ is the } v_j\text{th rule trace } \textit{clause} \text{ in that DNF, and} \tag{B.13}$$

$$X_{jv\rho} \text{ is the } \rho_{jv}\text{th condition } \textit{literal} \text{ in that rule trace.}$$

The explanation generation algorithms also apply to explanations for decision trees (see Section B.9), since we also convert decision trees to DNF.

B.5.1 NOTATION FOR EXPLANATIONS GIVEN INPUTS STATE

Given A as a Boolean expression that can be a literal or compound expression (conjunction, disjunction, DNF, etc.), and x as an inputs state, we introduce the notation

$$A|\mathrm{x} = \begin{cases} \mathrm{TRUE} & \text{, if } A \text{ is true given x} \\ \mathrm{TRUE} & \text{, if x} \notin A \\ \mathrm{FALSE} & \text{, otherwise} \end{cases}$$

where x ∉ A denotes that b is not described in A. In probability theory, this is equivalent:

$$[\![A|\mathrm{x}]\!] = \mathrm{P}(A|\mathrm{x}) = \begin{cases} 1 & \text{, if } A \text{ is true given x} \\ 1 & \text{, if x} \notin A \\ 0 & \text{, otherwise} \end{cases}$$

where for a deterministic, rule-based system, probability can only take values 0 or 1.

For an inference model, the ith class is inferred when

$$P(Y_i|x) \geq P(Y_j|x) \quad , \forall j \tag{B.14}$$

For a rule-based inference model inferring the ith class, each Y_j is mutually exclusive

$$P(Y_j|x) = [\![Y_j|x]\!] = \begin{cases} 1 & \text{, if } j = i \\ 0 & \text{, if } j \neq i \end{cases} \tag{B.15}$$

Conversely, the negation of Y_j yields the behavior:

$$[\![\bar{Y}_j|x]\!] = \begin{cases} 1 & \text{, if } j \neq i \\ 0 & \text{, if } j = i \end{cases} \tag{B.16}$$

B.5.2 CERTAINTY EXPLANATION

While it is possible to model rules with uncertainty (*e.g.*, Fuzzy control systems with fuzzy logic rules), currently, the rules of the Enactor framework does not provide certainty information.

B.5.3 WHY EXPLANATION

This explains why the ith class was inferred given the inputs x. We select a subset of the DNF tree can infer the ith class, specifically, the traces that do infer the ith class given inputs x:

$$\psi_i|x = \bigcup\nolimits_{v_i} \xi_{iv} [\![\xi_{iv}|x]\!] = \bigcup\nolimits_{v_i} \bigcap\nolimits_{\rho_{vk}} X_{jv\rho} [\![X_{iv\rho}|x]\!] \tag{B.17}$$

where

$$\xi_{iv} = \bigcap\nolimits_{\rho_{iv}} X_{iv\rho} \text{ is a trace consisting of conditions } X_{iv\rho}$$

$$[\![\xi_{iv}|x]\!] = \begin{cases} 1 & \text{, if } \xi_{iv} \text{ is true given inputs } x \\ 0 & \text{, otherwise} \end{cases}$$

$$\xi_{iv}|x = \bigcap\nolimits_{\rho_{iv}} X_{iv\rho}|x = \bigcap\nolimits_{\rho_{iv}} \bigcap\nolimits_{r=1}^{n} X_{iv\rho}|x_r \text{ is whether trace } \xi_{iv} \text{ is satisfied by inputs } x$$

$$X_{iv\rho}|x_r = \begin{cases} \text{TRUE} & \text{, if } X_{iv\rho} \text{ is true given } x_r \\ \text{TRUE} & \text{, if } x_r \notin X_{iv\rho} \\ \text{FALSE} & \text{, otherwise} \end{cases} \text{, is whether condition } X_{iv\rho} \text{ is satisfied by inputs } x$$

We can also notate Equation (B.56) in terms of individual conditions:

$$\psi_i|x = \bigcup\nolimits_{v_i, \xi_{iv}|x} \bigcap\nolimits_{\rho_{vk}} X_{jv\rho} \tag{B.18}$$

where $(v_i,\ \xi_{iv}|x)$ denotes traces that are satisfied by the inputs state x.

B.5.4 WHY NOT EXPLANATION

Given that the ith class was inferred, this provides an explanation for why the jth class was not inferred. First, we obtain traces that can infer the jth class under the relevant (not the current) input conditions. This selects all the traces in the DNF tree for the jth class, $\psi_j|x = \psi_j = \bigcup_{v_j} \xi_{jv}$. Furthermore, the negation of this DNF tree is true given inputs x and this is equivalent to a conjunctive normal form (CNF) using to De Morgan's laws:

$$\begin{aligned} \overline{\psi_j}\Big|x &= \left(\overline{\bigcup_{v_j} \xi_{jv}}\right)\Big|x \\ &\equiv \bigcap_{v_j} \overline{\xi_{jv}}|x = \bigcap_{v_j} \overline{\xi_{jv}}[\![\overline{\xi_{jv}}|x]\!] \end{aligned} \tag{B.19}$$

We define $\xi_{jv}|x$ to indicate the *filtered trace* of $\xi_{jv} = \cap_{\rho_{jv}} X_{jv\rho}$ that only includes conditions, $X_{jv\rho}$, that were satisfied by x. For a trace that is fully satisfied by x, where $[\![\xi_{jv}|x]\!] = 1$, this returns the full trace, *i.e.*,

$$\xi_{jv}|x = (\xi_{jv}|x)[\![\xi_{jv}|x]\!] = \xi_{jv}[\![\xi_{jv}|x]\!] = \xi_{jv} \tag{B.20}$$

Now, we consider the case where ξ_{jk} is not satisfied by x, *i.e.*,

$$[\![\xi_{jv}|x]\!] = 0 \quad \equiv \quad [\![\overline{\xi_{jv}}|x]\!] = 1$$

Note that while ξ_{jv} is a conjunction, $\overline{\xi_{jk}}$ is a disjunction as derivable by De Morgan's laws:

$$\overline{\xi_{jv}} = \overline{\bigcap_{\rho_{jv}} X_{jv\rho}} \equiv \bigcup_{\rho_{jv}} \overline{X_{jv\rho}}$$

$\overline{\xi_{jk}}|x$ indicates the *partial disjunction* of conditions, $\overline{X_{jv\rho}}$, that are satisfied by x, or conversely, the *partial* disjunction of conditions, $X_{jv\rho}$, that are not satisfied by x:

$$\overline{\xi_{jv}}|x = \bigcup_{\rho_{jv}} \overline{X_{jv\rho}}[\![\overline{X_{jv\rho}}|x]\!] \tag{B.21}$$

where

$$[\![X_{j\rho}|x]\!] = 0 \quad \equiv \quad [\![\overline{X_{j\rho}}|x]\!] = 1$$

For the Why Not explanation, we perform this treatment for all traces, ξ_{jv}, to get an expression in CNF:

$$\overline{\psi_j}|x = \bigcap\nolimits_{v_j} \overline{\xi_{jv}}|x = \bigcap\nolimits_{v_j} \bigcup\nolimits_{\rho_{jv}} \overline{X_{jv\rho}} [\![\overline{X_{jv\rho}}|x]\!] \tag{B.22}$$

where

$$\overline{X_{jv\rho}}|x = \begin{cases} \text{FALSE} & \text{, if } X_{iv\rho} \text{ is true given } x \\ \text{TRUE} & \text{, if } x_r \notin X_{iv\rho} \\ \text{TRUE} & \text{, otherwise} \end{cases}$$

Should purge duplicates.

B.5.5 HOW TO EXPLANATION

Suppose that the user is interested to know how the ith class may be inferred. The How To explanation returns the full DNF tree that can infer ith class:

$$\psi_i = \bigcup\nolimits_{j=1}^{m} \psi_j [\![i = j]\!] = \bigcup\nolimits_{v_i} \xi_{iv} = \bigcup\nolimits_{v_i} \bigcap\nolimits_{\rho_{vk}} X_{jv\rho} \tag{B.23}$$

Essentially, this explanation describes the various conditions that must be true for the ith class to be inferred, either one of the rule traces ξ_{iv}.

Explanation algorithms for rule DNF is implemented in the explainer delegate `DnfTreeExplainerDelegate`.

B.6 SEPARABLE-RULES DNF EXPLAINER

We may want to retain some semantic meaning in individual rules, rather than naively merge them together. For example, rules may be given names to aid end-users to remember their purpose or rationale, and to help them distinguish between other rules.

For an inference model with m possible class values, we will store a map of DNF trees for each class value, j, in a *map of maps* of DNF trees:

$$\Psi = \bigcup_{j=1}^{m} \psi_j \tag{B.24}$$

where

$$\psi_j = \bigcup_{k_j} \psi_{jk} \text{ is the map of all DNF trees for all rules that infer the } j\text{th class,} \tag{B.25}$$

$$\psi_{jk} = \bigcup_{v_{jk}} \xi_{jkv} \text{ is the } k_j\text{th rule as a DNF tree,} \tag{B.26}$$

$$\xi_{jkv} = \bigcap_{\rho_{jkv}} X_{jkv\rho} \text{ is the } v_{jk}\text{th rule trace } \textit{clause} \text{ in that DNF, and} \tag{B.27}$$

$$X_{jkv\rho} \text{ is the } \rho_{jkv}\text{th condition } \textit{literal} \text{ in that rule trace.}$$

B.6.1 WHY EXPLANATION

Substituting Equation(B.24) into (B.17) gives:

$$\psi_i|x = \bigcup_{k_i} \bigcup_{v_{ik}} \xi_{ikv} [\![\xi_{iv}|x]\!] \tag{B.28}$$

which is a double disjunction of trace clauses, where traces due to each v_{ik}th rule is kept separate.

For the simple case where each rule has only one trace conjunction in its DNF form, *i.e.*, $|\psi_{jk}| = 1$, Equation (B.28) is simplified to

$$\psi_i|x = \bigcup_{k_i} \xi_{ik} [\![x \vdash \xi_{ik}]\!] \tag{B.29}$$

So, the explanation only provides a disjunction of satisfied traces.

B.6.1.1 WHY EXPLANATION OF MERGED RULES

Merging results from Equation (B.28) yields:

$$\psi_i|x = \bigcup_{k_i, v_{ik}} \xi_{ikv} [\![\xi_{iv}|x]\!] \tag{B.30}$$

Note that this only adds satisfied (*i.e.*, $x \vdash \xi_{ikv}$) traces from each DNF tree to the explanation, not the whole DNF tree.

B.6.1.2 WHY EXPLANATION AS RULE NAME

This presents the same information as Equation (B.28), but allows incremental retrieval of explanations. First, we retrieve the rule by name

$$\bigcup_{v_i} v_i [\![\xi_{iv}|x]\!] \tag{B.31}$$

where v_i refers to the index or name for the v_ith rule.

Later, the end-user may ask for details about that rule, and can see explanation:

$$\zeta_{iv} = \bigcup_{u_i} \xi_{iu} [\![v = u]\!] = \xi_{iv}$$

Programmatically, this is equivalent to $\Psi.\mathrm{getDNFs}(i).\mathrm{getDNF}(v)$.

B.6.2 WHY NOT EXPLANATION

Substituting Equation (B.24) into (B.22) gives a conjunction of CNFs:

$$\overline{\psi_J}|x = \bigcap_{k_j} \bigcap_{v_k} \overline{\xi_{Jkv}}|x \equiv \bigcap_{k_j} \bigcap_{v_{jk}} \bigcup_{\rho_{jkv}} \overline{X_{Jkv\rho}} [\![\overline{X_{Jkv\rho}}|x]\!] \tag{B.32}$$

B.6.3 HOW TO EXPLANATION

This is similar to Equation (B.23), but we retain separation of each DNF tree representing a different rule:

$$\psi_i = \bigcup_{j=1}^{m} \psi_j [\![i = j]\!] = \bigcup_{k_i} \bigcup_{v_i} \xi_{iv} \tag{B.33}$$

B.7 ENACTOR RULES EXPLAINER

This implements the algorithms in Rule Trace Explainer (see Section B.4) to explain the rules executed in Enactors [Dey and Newberger, 2009] within the Context Toolkit [Dey, Abowd, and Salber]. First, we parse the rule encoded as `AbstractQueryItems` into the `Expression` framework of the Intelligibility Toolkit. We then apply algorithms in Figure B.2 and Figure B.3 to generate the map of DNF trees, Ψ, allowing us to generate explanations in Section B.5.

B.8 OTHER SYSTEM RULES EXPLAINERS

Other context-aware systems that use rules for inference may also be able to leverage the algorithms in Sections B.4 and B.5 for explaining rules by parsing their rules into into the `Expression` framework of the Intelligibility Toolkit.

B.9 DECISION TREE RULE TRACE EXPLAINER

We parse the decision tree structure into disjunctive normal form (DNF), from which we can more easily generate explanations. Each conjunction represents a trace from the root node of the tree to a leaf. Therefore, the number of conjunctions is equal to the number of leaves.

$$\begin{gathered} \tau \xrightarrow{\text{to DNF}} \psi \\ \psi = \text{TreeToDNF}(\tau) \end{gathered} \tag{B.34}$$

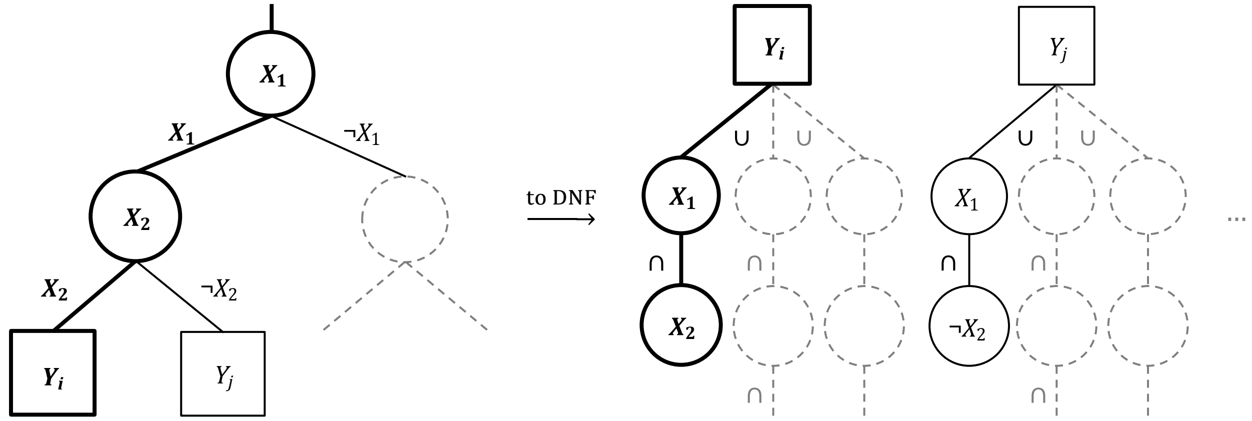


Figure B.4. (Left) A simple decision tree illustrating the condition literal of each edge and the class value that will be inferred at each leaf. (Right) DNF trees for each class value that is built after traversing the tree and collecting traces with the same inferred class value at their leaves. Although this diagram illustrates the conversion for a binary decision tree, the conversion is applicable to trees with more than two branches at each node.

Figure B.4 illustrates the decision tree conversion to DNF, and Figure B.5 and Figure B.6 describes the algorithm. Once in DNF, we generate explanations using techniques in Section B.5.

```
$\tau_0$ = root node of decision tree
$\xi$ := new empty trace clause // stores a conjunction of condition literals, $X_v$
$\Psi = \bigcup_{j=1}^{m} \psi_j$ = map of DNFs of each class value // gets updated in TreetoDNF
TreeToDNF($\tau_0, \xi, \Psi$)

TreeToDNF($\tau, \xi, \Psi$)
    If $\tau$ parent of $\sum_v \tau_v$ for some $\tau_v$ // $\tau$ has child nodes
        For each child node $\tau_v$
            $X_v$ := EdgeLiteral($\tau, \tau_v$) // define condition literal to represent edge $\{\tau, \tau_v\}$ from $\tau$ to $\tau_v$
            $\xi_v$ := Clone $\xi$ // duplicate trace to extend for each child path
            $\xi_v := \xi_v + X_v$ // append literal $X_v$ to trace clause $\xi_v$
            TreeToDNF($\tau_v, \xi_v, \Psi$)
    Else // $\tau$ is a leaf
        $P_\tau = \sum_{j=1}^{m} P_{\tau,j}$ := read probability class distribution at node $\tau$
        $i$ := $\max_j P_{\tau,j}$ // inferred class value has the maximum class probability
        $\psi_i = \Psi \cdot [\![j = i]\!]$ // retrieve DNF $\psi_i$ that infers the $i$th class
        $\psi_i := \psi_i + \xi$ // append trace clause $\xi$ to $\psi_i$
```

Figure B.5. Algorithm to traverse the decision tree depth-first to convert it into DNF.

```
EdgeLiteral($\tau, \tau_v$)
    $x_r$ = attribute described in $\tau$
    If $x_r$ is nominal // $\tau$ describes a nominal attribute split
        $X_v$: $x_r = \rho$th value of $x_r$
    Else If $x_r$ is numeric // $\tau$ describes a numeric attribute binary split
        If $v = 0$
            $X_v$: $x_r \le x_{r,split}$ // $x_{r,split}$ = learned split point for $r$th feature at this node $\tau$
        Else If $v = 1$
            $X_v$: $x_r > x_{r,split}$
    Return $X_v$
```

Figure B.6. Algorithm to obtain the condition literal representing an edge transition in the decision tree. This is specifically implemented for J48 in Weka, but can be generally applicable for other decision tree implementations.

This algorithm is implemented in `J48RuleTraceExplainer`.

B.10 Other Rules Explainers

Other rule explainers may also be developed. For example, a more efficient algorithm to convert an arbitrary Boolean expression to DNF is the EXPAND algorithm proposed in [Crama and Hamer, 2008, p. 22], which was inspired by Tseitin [1968] and [Blair, Jeroslow, and Lowe, 1986]. This algorithm runs in polynomial time instead of exponential time of the previously described naïve method. This can be implemented in a rule explainer to speed up the explanation generation.

Aules can be extracted from inference models that are intrinsically not rule-based, such as Artificial Neural Networks (ANN) [Tickle *et al.*, 1998], and Support Vector Machines (SVM) [Núñez, H., Angulo, C., Català, 2002]. This allows other non-rule-based models to be explained using the rule paradigm, which is familiar to lay end-users.

B.11 Weights of Evidence Explainer

This explainer forms the abstract basis of our subsequent weights of evidence explainers. It uses the underlying concept that an inference is due to a total evidence, where this evidence may support or oppose the inference, and can be due to a *sum* of underlying *atomic* weights of evidence. These weights may be due to the input feature values voting for or against an inference. Depending on the inference model, there may also be more *dimensions* of atomic weights. For simplicity, we denote an atomic evidence as f_{ir}, and the space of all atomic weights as R. A total evidence can thus be represented as the sum:

$$g_i = \sum\nolimits_{r \in R} f_{ir} \tag{B.35}$$

Equation (B.35) requires that the explainer is able to derive a *linear additive* expression of atomic units. This is easy for linear classifiers (*e.g.*, linear SVM), but in general, isotonic (monotonic increasing) transformations may be required (*e.g.*, see explainer for naïve Bayes in Section B.15). We defer these steps to later sections describing the concrete explainers.

We shall next show how with this *absolute* weights of evidence, we can derive weights of evidence explanations for Why and Why Not questions.

B.11.1 ABSOLUTE EVIDENCE

This explains the value of p_i as a total evidence due to the sum of atomic weights:

$$g_i = \sum\nolimits_{r} f_{ir} \tag{B.36}$$

where f_{ir} is the rth atomic weight of evidence.

B.11.2 WHY NOT EXPLANATION

This explains why the jth class was *not* inferred over the ith class. In other words, why the ith class was inferred instead of the jth class.

$$p_i \geq p_j$$

Note that this is different from the Why Not explanation of a *rule trace* that explains why the jth class was not inferred. In this case, the jth class may have been inferred, but just not with the highest certainty among all class values.

$$\Delta g_{ij} = g_i - g_j = \sum\nolimits_{r} \Delta f_{ijr} \geq 0 \tag{B.37}$$

where $\Delta f_{ijr} = f_{ir} - f_{jr}$ and we assume that the atomic weights of evidence are separable by each atomic unit.

B.11.3 WHY EXPLANATION

This explains why the ith class was inferred over all other class values j, *i.e.*,

$$p_i \geq p_j \quad , \forall j$$

Consequently, Equation (B.37) holds for $\forall j$, such that we can sum over $\forall j$ to get the Why explanation:

$$\Delta g_{i\forall} = \sum_{j=1}^{m} \Delta g_{ij} = \sum_{j=1}^{m} \sum_{r} \Delta f_{ijr} \geq 0 \tag{B.38}$$

B.11.4 CERTAINTY EXPLANATION

Assuming that at the system level, the model will produce a distribution of certainty for inferring each class value, the Certainty explanation returns this distribution:

$$p = \sum_{j=1}^{m} p_j \tag{B.39}$$

where p is the total certainty (usually normalized as a probability to 1) and p_j is the certainty for inferring the jth class value.

B.11.5 HOW TO EXPLANATION

The How To explanation depends on underlying model and we defer this to the later sections. Also note that since this is a general explanation independent of specific instances, we do not provide the explanation in terms of input feature values, x_r. Generally, this explanation will be presented as a multivariate inequality with the input features as variables.

Consider that the ith class was inferred and the user asks How To infer the hth class. First we compute the current evidence for the hth class using Equation (B.36):

$$g_h = \sum_{r} f_{hr}$$

Substituting this into Equation (B.38), we get

$$\Delta g_{h\forall}(\boldsymbol{x}) = \sum_{j=1}^{m} \Delta g_{hj}(\boldsymbol{x}) = \sum_{j=1}^{m} \sum_{r} \Delta f_{hjr}(x_r) \geq 0 \tag{B.40}$$

where $\Delta g_{hj}(\boldsymbol{x})$ is separable for each x_r term and $\Delta f_{hjr}(x_r)$ is a function of only the x_r term. The exact expression of $\Delta f_{hjr}(x_r)$ depends on the Explainer. We describe three common forms.

B.11.5.1 NUMERIC LINEARLY SEPARABLE

Some Explainers (*e.g.*, for linear regression, logistic regression, SVM, kNN) produce expressions for $\Delta g_{hj}(\boldsymbol{x})$ which are linearly separable for each x_r, *i.e.*,

$$\begin{aligned} \Delta g_{h\forall}(\boldsymbol{x}) &= \sum_{j=1}^{m} \sum_{r} \Delta f_{hjr}(x_r) = \sum_{j=1}^{m} \sum_{r} \Delta \varpi_{hjr} x_r \geq 0 \\ &= \sum_{r} \left(\sum_{j=1}^{m} \Delta \varpi_{hjr} \right) x_r \end{aligned} \tag{B.41}$$

where

$$\Delta f_{hjr}(x_r) = \Delta \varpi_{hjr} x_r \tag{B.42}$$

is a linear function of x_r and $\Delta \varpi_{hjr}$ is a coefficient, of which concrete Explainers will need to derive expressions (or approximations).

The How To explanation expresses a linear equation of input feature values, x_r such that $\Delta g_{h\forall} \geq 0$. For specific How To explanations of each Weights of Evidence explainer, we will just derive expressions for $\Delta g_{h\forall}(\boldsymbol{x})$ in terms of input features, $\boldsymbol{x} = (x_1, \ldots, x_n)$.

B.11.5.2 NUMERIC NORMAL DISTRIBUTION

Some Explainers (*e.g.*, for naïve Bayes) assume that numeric attributes follow a Normal distribution given the class value, i, *i.e.*,

$$P(x_r|i) = p_{ir} = \frac{1}{\sqrt{2\pi\sigma_{ir}^2}} e^{-\frac{(x_r - \mu_{ir})^2}{2\sigma_{ir}^2}} \tag{B.43}$$

$$\log(p_{ir}) = -\frac{1}{2}\log(2\pi) - \log(\sigma_{ir}) - \frac{(x_r - \mu_{ir})^2}{2\sigma_{ir}^2} \tag{B.44}$$

where μ_{ir} is the mean value of the rth feature and σ_{ir} is its standard deviation.

An approximate How To explanation for inferring the hth class is just to return the mean value μ_{hr} of each rth input feature which maximizes $P(x_r|h)$, *i.e.*,

$$\underset{x_r}{\operatorname{argmax}}(p_{ir}) = \mu_{ir} \tag{B.45}$$

B.11.5.3 NOMINAL INPUT FEATURE

If each x_r input feature is a nominal / discrete variable, then we can find all combinations of $\boldsymbol{x}$ such that $\Delta g_{h\forall}(\boldsymbol{x}) \geq 0$ through permutation.

B.11.6 HOW TO IF EXPLANATION

We can provide a more operationally useful explanation with How To If that fixes (makes constant) all but one input feature, $x_{\tilde{r}}$. We consider the current total evidence for inferring the hth class, $\Delta g_{h\forall}(\hat{\boldsymbol{x}})$, where the hat notation indicates current input feature values of the instance, $\hat{\boldsymbol{x}}$. Note that the actual value of $\Delta g_{h\forall}(\hat{\boldsymbol{x}})$ depends on whether the hth class was actually inferred, *i.e.*,

$$\Delta g_{h\forall}(\hat{\boldsymbol{x}}) \begin{cases} \geq 0 & \text{, if } h = i \\ < 0 & \text{, if } h \neq i \end{cases}$$

The How To If explanation describes the range of values of $x_{\tilde{r}}$ to satisfy the condition $\Delta g_{h\forall} \geq 0$, *i.e.*:

$$\Delta g_{h\forall}(\hat{\boldsymbol{x}}) + \sum_{j=1}^{m} \left(\Delta f_{hjr}(x_{\tilde{r}}) - \Delta f_{hjr}(\hat{x}_{\tilde{r}})\right) \geq 0 \tag{B.46}$$

Similarly to the How To explanation, the How To If depends on the form of the input feature.

B.11.6.1 NUMERIC LINEARLY SEPARABLE

Substituting Equation (B.42) into Equation (B.46) gives the How To If explanation:

$$\Delta g_{h\forall}(\hat{\boldsymbol{x}}) + \left(\sum_{j=1}^{m} \Delta\omega_{hjr}\right)(x_{\tilde{r}} - \hat{x}_{\tilde{r}}) \geq 0$$

$$x_{\tilde{r}} \begin{cases} > x^{\dagger}_{\tilde{r}} & \text{, if } \sum_{j=1}^{m} \Delta\omega_{hjr} > 0 \\ < x^{\dagger}_{\tilde{r}} & \text{, if } \sum_{j=1}^{m} \Delta\omega_{hjr} < 0 \\ \emptyset & \text{, otherwise} \end{cases} \tag{B.47}$$

where

$$x^{\dagger}_{\tilde{r}} = \hat{x}_{\tilde{r}} - \frac{\Delta g_{h\forall}(\hat{\boldsymbol{x}})}{\sum_{j=1}^{m} \Delta\omega_{hjr}}$$

B.11.6.2 NUMERIC NORMAL DISTRIBUTION

Considering atomic weights of evidence as log transformations of probabilities (*e.g.*, Equation (B.93)), we have:

$$\begin{aligned} \Delta f_{hjr}(x_r) &= f_{hr}(x_r) - f_{jr}(x_r) = \log(p_{hr}) - \log(p_{jr}) \\ &= \log(\sigma_{jr}) + \frac{(x_r - \mu_{jr})^2}{2\sigma_{jr}^2} - \log(\sigma_{hr}) - \frac{(x_r - \mu_{hr})^2}{2\sigma_{hr}^2} \end{aligned} \tag{B.48}$$

EQUAL VARIANCE

Assuming equal variances, $\sigma_{hr} \approx \sigma_{jr}$, Equation (B.48) becomes

$$\begin{aligned} \Delta f_{hjr}(x_r) &= \log(\sigma_{hr}) + \frac{(x_r - \mu_{jr})^2}{2s_r^2} - \log(\sigma_{hr}) - \frac{(x_r - \mu_{hr})^2}{2s_r^2} \\ &= \frac{(x_r - \mu_{jr})^2 - (x_r - \mu_{hr})^2}{2\sigma_{hr}^2} = \frac{(\mu_{hr} - \mu_{jr})(2x_r - \mu_{jr} - \mu_{hr})}{2\sigma_{hr}^2} \\ &= \frac{\mu_{hr} - \mu_{jr}}{\sigma_{hr}^2}\left(x_r - \frac{\mu_{hr} + \mu_{jr}}{2}\right) = A_{hjr}(x_r - x^{*}_{hjr}) \end{aligned} \tag{B.49}$$

where

$$A_{hjr} = \frac{\mu_{hr} - \mu_{jr}}{\sigma_{hr}^2} \quad \text{and} \quad x^{*}_{hjr} = \frac{\mu_{hr} + \mu_{jr}}{2}$$

Therefore, substituting Equation (B.49) into Equation (B.46):

$$\Delta g_{h\forall}(\widehat{\boldsymbol{x}}) + \sum_{j=1}^{m} A_{hj\tilde{r}}(x_{\tilde{r}} - \hat{x}_{\tilde{r}}) \geq 0 \tag{B.50}$$

Solving Equation (B.50), we get the range of values of $x_{\tilde{r}}$ for the How To If explanation:

$$x_{\tilde{r}} \begin{cases} > x_{\tilde{r}}^{\dagger} & \text{, if } \sum_{j=1}^{m} A_{hj\tilde{r}} > 0 \\ < x_{\tilde{r}}^{\dagger} & \text{, if } \sum_{j=1}^{m} A_{hj\tilde{r}} < 0 \\ \emptyset & \text{, otherwise} \end{cases} \tag{B.51}$$

where

$$x_{r}^{\dagger} = \hat{x}_{\tilde{r}} - \frac{\Delta g_{h\forall}(\widehat{\boldsymbol{x}})}{\sum_{j=1}^{m} A_{hj\tilde{r}}} = \hat{x}_{\tilde{r}} - \frac{2\Delta g_{h\forall}(\widehat{\boldsymbol{x}})}{m\mu_{hr} - \sum_{j=1}^{m} \mu_{jr}}$$

and $\sum_{j=1}^{m} A_{hj\tilde{r}} > 0$ when $\mu_{h\tilde{r}} > \frac{1}{m}\sum_{j=1}^{m} \mu_{j\tilde{r}}$.

UNEQUAL VARIANCE

We can also solve for a range of values for $x_{\tilde{r}}$ without assuming equal variance and, instead, solve a quadratic equation. Equation (B.48) becomes:

$$\begin{aligned} \Delta f_{hjr}(x_r) &= \frac{1}{2}\left(\frac{1}{\sigma_{jr}^2} - \frac{1}{\sigma_{hr}^2}\right)x_r^2 - \left(\frac{\mu_{jr}}{\sigma_{jr}^2} - \frac{\mu_{hr}}{\sigma_{hr}^2}\right)x_r + \left(\left(\frac{\mu_{jr}^2}{\sigma_{jr}^2} - \frac{\mu_{hr}^2}{\sigma_{hr}^2}\right) + \log\left(\frac{\sigma_{jr}}{\sigma_{hr}}\right)\right) \\ &= A_{hjr}x_r^2 + B_{hjr}x_r + C_{hjr} \end{aligned} \tag{B.52}$$

where

$$A_{hjr} = \frac{1}{2}\left(\frac{1}{\sigma_{jr}^2} - \frac{1}{\sigma_{hr}^2}\right), \quad B_{hjr} = \frac{\mu_{jr}}{\sigma_{jr}^2} - \frac{\mu_{hr}}{\sigma_{hr}^2}, \quad C_{hjr} = \left(\frac{\mu_{jr}^2}{\sigma_{jr}^2} - \frac{\mu_{hr}^2}{\sigma_{hr}^2}\right) + \log\left(\frac{\sigma_{jr}}{\sigma_{hr}}\right)$$

Substituting Equation (B.52) into Equation (B.46):

$$\begin{gathered}\left(\sum_{j=1}^{m} A_{hj\tilde{r}}\right) x_{\tilde{r}}^{2} + \left(\sum_{j=1}^{m} B_{hj\tilde{r}}\right) x_{\tilde{r}} + \Delta g_{h\forall}(\hat{\boldsymbol{x}}) \geq 0 \\ A_{h\tilde{r}} x_{\tilde{r}}^{2} + B_{h\tilde{r}} x_{\tilde{r}} + C_{h} \geq 0\end{gathered} \tag{B.53}$$

where

$$A_{h\tilde{r}} = \sum_{j=1}^{m} A_{hj\tilde{r}}\,, \quad B_{h\tilde{r}} = \sum_{j=1}^{m} B_{hj\tilde{r}}\,, \quad C_{h} = \Delta g_{h\forall}(\hat{\boldsymbol{x}})$$

Solving Equation (B.53), we get the range of values of $x_{\tilde{r}}$ for the How To If explanation:

$$x_{\tilde{r}}\colon \begin{cases} \forall x_{\tilde{r}} & \text{, if } A_{h\tilde{r}} \geq 0, C_{h} \geq 0 \\ x_{\tilde{r}} < x_{\tilde{r}}^{(-)} \text{ or } x_{\tilde{r}} > x_{\tilde{r}}^{(+)} & \text{, if } A_{h\tilde{r}} \geq 0, C_{h} < 0 \\ x_{\tilde{r}}^{(-)} < x_{\tilde{r}} < x_{\tilde{r}}^{(+)} & \text{, if } A_{h\tilde{r}} < 0, C_{h} \geq 0 \\ \emptyset & \text{, if } A_{h\tilde{r}} < 0, C_{h} < 0 \end{cases} \tag{B.54}$$

where

$$x_{\tilde{r}}^{(-)} = \frac{-B_{h\tilde{r}} - \sqrt{B_{h\tilde{r}}^{2} - 4A_{h\tilde{r}}C_{h}}}{2A_{h\tilde{r}}}, \quad x_{\tilde{r}}^{(+)} = \frac{-B_{h\tilde{r}} + \sqrt{B_{h\tilde{r}}^{2} - 4A_{h\tilde{r}}C_{h}}}{2A_{h\tilde{r}}}$$

B.12 LINEAR REGRESSION EXPLAINER

A common and simple way to model application behavior is using a linear function, where multiple input factors influence an output value (*e.g.*, outdoor temperature and number of occupants influencing the heat output of a temperature control system). Formally, the function takes the form:

$$y = a_{0} + a_{1}x_{1} + a_{2}x_{2} + \cdots + a_{n}x_{n} = \sum_{r=0}^{n} a_{r}x_{r} \tag{B.55}$$

where a_{r} is the scaling factor for each input, and x_{r} is the value of the rth input feature.

For a given input state, $\{x_r\} = \langle x_1, x_2, \dots, x_n \rangle$, we can explain how each input feature value, x_r, influences the output due to the value $f_r = a_r x_r$. Thus we can explain the outcome using the Weights of Evidence explanation:

$$g = f_0 + f_1 + f_2 + \cdots + f_n = \sum\nolimits_{r=0}^{n} f_r \tag{B.56}$$

where $f_r = a_r x_r$ is the evidence due to each input feature, and g is the total evidence.

Equation (B.56) forms the basis of the remaining Weights of Evidence explanations in terms of input features.

B.12.1 CERTAINTY EXPLANATION AS MEASUREMENT UNCERTAINTY

While the most classification techniques covered in the Intelligibility Toolkit models the certainty (or confidence) of inferring the output as various nominal class values, certainty in regression is a different concept typically indicating the uncertainty or error of the numeric output. This is particularly meaningful for inputs representing physical measures (*e.g.*, temperature, energy). Using the method *addition in quadrature* that calculates uncertainty due to sums and differences for random and independent uncertainties, the uncertainty of the output is:

$$\delta y = \sqrt{(\delta a_0)^2 + \big(\delta(a_1 x_1)\big)^2 + \cdots + \big(\delta(a_n x_n)\big)^2} = \sqrt{\sum\nolimits_{r=0}^{n} \big(\delta(a_r x_r)\big)^2} \tag{B.57}$$

where δz represents the error of the term z, specifically, δa_r is the uncertainty in a_r, and δx_r is the uncertainty in x_r. δx_r could be the conditional standard deviation or a confidence interval given the inferred value$y$, or the actual measurement error in x_r. For manually defined functions, each δa_r could be 0 or a constant value; for learned regression functions, δa_r could be the least squares error.

Assuming independence between the estimated parameter a_r and stochastic variable x_r, there will be no covariance or correlation between them, and Equation (B.57) becomes:

$$\delta y = \sqrt{\sum\nolimits_{r=0}^{n} \big(\delta(a_r x_r)\big)^2} = \sqrt{\sum\nolimits_{r=0}^{n} (\delta a_r x_r + a_r \delta x_r)^2} \tag{B.58}$$

For simplicity, if we assume no uncertainty in a_r, Equation (B.57) then becomes:

$$\delta y = \sqrt{\sum\nolimits_{r=0}^{n} (a_r \delta x_r)^2} \tag{B.59}$$

Note that this results represents the measurement *un*certainty in the output value, not the certainty that the output value is correct.

B.13 LOGISTIC REGRESSION EXPLAINER

Now that we have dealt with explaining functional outputs, we turn to explaining functions that are used for pairwise categorical inference (two categorical states). In particular, Logistic Regression is commonly used to transform weights output into a choice between two values (*i.e.*, 0 or 1). We begin by considering the simpler case of logistic regression for binary classification and then discuss it for the multiclass problem.

B.13.1 BINARY LOGISTIC REGRESSION

The binary Logistic Regression models the probability of inferring class value 1 (instead of 0) as a sigmoid function of a linear expression of input features:

$$p = \frac{1}{1 + e^{-\sum_{r=0}^{n} \beta_r x_r}} \tag{B.60}$$

where x_r is the value for the rth input feature, β_r is the *gain*, and $x_0 = 1$ is constant term.

A logit (log-odds) transform, $\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$, on Equation (B.60) converts it to linear form:

$$\text{logit}(p) = \sum\nolimits_{r=0}^{n} \beta_r x_r \tag{B.61}$$

Equation (B.61) is exactly the same form as Equation (B.55) for linear regression, and so we can apply the same explanation techniques as for linear regression:

$$g = \sum\nolimits_{r=0}^{n} f_r \tag{B.62}$$

where $f_r = \beta_r x_r$ is the evidence due to each input feature, and $g = \text{logit}(p)$ is the total evidence.

B.13.2 MULTINOMIAL LOGISTIC REGRESSION

Logistic Regression is intrinsically a binary classifier, so a multi-class extension approach needs to be applied to support multiclass classification. The Weka toolkit uses a one-*vs.*-one approach and models a multinomial Logistic Regression with m class values and n input features with:

- $m-1$ pairwise classifiers are built that compares the first $m-1$ class values against the last class. The ith classifier learns the linear function
$$y_i(x) = \sum_{r=0}^{n} \beta_{ir} x_r$$
- The probability of inferring the ith class value $(i < m)$ for the current input state is
$$p_i = \frac{e^{y_i(x)}}{1 + Y_\Sigma(x)} = \frac{1}{Z} e^{y_i(x)}$$
- The probability of inferring the mth class value (*i.e.*, the last class value) is
$$p_m = \frac{1}{1 + Y_\Sigma(x)} = \frac{1}{Z} e^{y_m(x)}$$

where $Z = 1 + Y_\Sigma(x)$ is the normalization factor, $Y_\Sigma(x) = \sum_{j=1}^{m-1} e^{y_j(x)}$, $e^{y_m(x)} = 1$, and $y_m(x) = \sum_{r=0}^{n} \beta_{mr} x_r = 0$. We choose $\beta_{mr} = 0, \forall r, \forall x_r$. We can see that this satisfies $\sum_{i=1}^{m} p_i = 1$.

Unfortunately, the total evidence for inferring the mth class is 0, and this can lead to problems when we want to use this evidence to explain ensemble methods (see later, Section B.20), because they may need to normalize a non-zero total evidence. We can work around this problem by relaxing the requirement that $y_m(x) = 0$, and defined $y_m(x) = \sum_{r=0}^{n} \beta_{mr} x_r = \beta_{m0}$, where

$$\beta_{mr} = \begin{cases} \beta_{m0} & r = 0 \\ 0 & r > 0 \end{cases}, \forall r, \forall x_r$$

For simplicity, we can set $\beta_{m0} = 1$. We also need to preserve the proportionality among each p_i, *i.e.*:

$$\frac{p_m}{p_i} = e^{y_m(x) - y_i(x)} = e^{\beta_{m0} - (y_i(x) + \beta_{m0})}$$

$$\frac{p_j}{p_i} = e^{y_j(x) - y_i(x)} = e^{(y_j(x) + \beta_{m0}) - (y_i(x) + \beta_{m0})}$$

So β_{m0} needs to be added to each pairwise classifier. This changes the probability of inferences:

1. $m-1$ pairwise classifiers are built that compares the first $m-1$ class values against the last class. The ith classifier learns the linear function
$$y_i(x) = \sum\nolimits_{r=0}^{n} \beta_{ir} x_r + \beta_{m0}$$
2. The probability of inferring the ith class value $(i < m)$ for the current input state is
$$p_i = \frac{e^{y_i(x)}}{e^{\beta_{m0}} + Y_\Sigma(x)} = \frac{1}{Z} e^{y_i(x)}$$
3. The probability of inferring the mth class value (*i.e.*, the last class value) is
$$p_m = \frac{1}{e^{\beta_{m0}} + Y_\Sigma(x)} = \frac{1}{Z} e^{y_m(x)}$$

where $Z = e^{\beta_{m0}} + Y_\Sigma(x)$ is the normalization factor, $Y_\Sigma(x) = \sum_{j=1}^{m-1} e^{y_j(x)}$, and $e^{y_m(x)} = e^{\beta_{m0}}$, $y_m(x) = \sum_{r=0}^{n} \beta_{mr} x_r = \beta_{m0}$. We can see that this still satisfies $\sum_{i=1}^{m} p_i = 1$.

To get a linear additive Weights of Evidence, we take a log transform:

$$\begin{aligned} g_i &= \mathrm{logit}(p_i) \equiv y_i(x) \\ &= \sum\nolimits_{r=0}^{n} f_{ir} \end{aligned} \tag{B.63}$$

where

$$f_{ir} = \beta_{ir} x_r$$

$$\beta_{ir} = \begin{cases} \beta_{ir} & i < m, r > 0 \\ \beta_{ir} + \beta_{m0} & i < m, r = 0 \\ 0 & i = m, r > 0 \\ \beta_{m0} & i = m, r = 0 \end{cases} , \beta_{m0} \neq 0$$

$$x_r = \begin{cases} x_r & r > 0 \\ 1 & r = 0 \end{cases}$$

B.13.3 INPUTS FEATURE NORMALIZATION AND STANDARDIZATION

Often it is useful to pre-process the data before training or using an inference model (*e.g.*, SMO, kNN) by performing normalization:

$$z_r = \frac{x_r - \mu_r}{\sigma_r} \tag{B.64}$$

or min-max normalization (called *standardization* in the Weka toolkit; scales the range to 0 to 1):

$$z_r = \frac{x_r - min_r}{max_r - min_r} \tag{B.65}$$

This can reduce bias due to the relative value sizes of different features. The consequence of this is that the model is trained and tested on the normalized/standardized features, which is not as meaningful to end-users as the original features. Particularly, this affects the weights used in the Weights of Evidence style of explanation. Therefore we need to invert these pre-processes in generated explanation. To invert normalization:

$$f_r = w_r z_r = w_r \frac{x_r - \mu_r}{\sigma_r} = \frac{w_r}{\sigma_r} x_r - \frac{w_r \mu_r}{\sigma_r}$$

$$\begin{aligned} g &= \sum_{r=1}^{n} f_r - b \\ &= \sum_{r=1}^{n} \left(\frac{w_r}{\sigma_r} x_r - \frac{w_r \mu_r}{\sigma_r} \right) - b \\ &= \sum_{r=1}^{n} w_r{}' x_r - b' = \sum_{r=1}^{n} f_r{}' - b' \end{aligned} \tag{B.66}$$

where μ_r is the mean value of the rth input feature, σ_r its standard deviation over the training dataset, and

$$f_r' = w_r{}' x_r = \frac{w_r}{\sigma_r} x_r$$

$$b' = \sum_{r=1}^{n} \frac{w_r \mu_r}{\sigma_r} + b$$

Similarly, to invert min-max normalization:

$$g = \sum_{r=1}^{n} f_r{}' - b' \tag{B.67}$$

where

$$f_r' = w_r{}' x_r = \frac{w_r}{max_r - min_r} x_r \quad , \quad b' = \sum_{r=1}^{n} \frac{w_r min_r}{max_r - min_r} + b$$

B.14 LINEAR SUPPORT VECTOR MACHINE (SVM) EXPLAINER

Linear support vector machines (SVM) use a maximum-margin hypothesis to train a linear classifier to discriminate between two (binary) class values. In its simplest form, it outputs a continuous numeric output, which can be calibrated to give a more accurate probability output for its inference. Several techniques exist to extend linear SVMs to handle multiple class values, uncalibrated or calibrated. We describe these algorithms and discuss explainer algorithms used in the Intelligibility Toolkit to explain these four variants of Linear SVMs.

B.14.1 BINARY NON-CALIBRATED LINEAR SVM

Even though linear SVMs use a different learning approach than linear or logistic regression (*e.g.*, Sequential Minimal Optimization (SMO) [Platt, 1998]), it produces a linear decision boundary that takes the same form, *i.e.*

$$y = \sum_{r=0}^{n} w_r x_r \tag{B.68}$$

where y is the functional output, x_r is the value of the rth input feature, w_r is the learned weight of the feature, and there are n features. We can thus apply the same Weights of Evidence approach as for Linear Regression to explain this linear decision boundary (Equation (B.56)).

$$g = \sum_{r=0}^{n} f_r \tag{B.69}$$

where $f_r = w_r x_r$.

B.14.2 BINARY PLATT-CALIBRATED LINEAR SVM

The simple linear SVM does not produce a probability output. Platt [1999] developed a calibration method by fitting the SVM output to a logistic regression with the following model:

$$p = \frac{1}{1 + e^{-(\beta y + \gamma)}} \tag{B.70}$$

where β and γ is the learned coefficient and constant, respectively, and y is the learned linear model from Equation (B.68).

Taking a logit transform on Equation (B.70) produces the similar expression as Equation (B.68) but with a scaling factor β and translation γ:

$$g = \sum_{r=0}^{n} \beta w_r x_r + \gamma = \sum_{r=0}^{n} f_r \tag{B.71}$$

where $f_r = \omega_r x_r$,

$$\omega_r = \begin{cases} \beta w_r & \text{, if } r > 0 \\ \beta w_0 + \gamma & \text{, if } r = 0 \end{cases}$$

B.14.3 MULTICLASS NON-CALIBRATED LINEAR SVM

We consider multiclass inference as done in the Weka toolkit for linear SVM. The number of votes for the ith class value is the sum of relevant pairwise classifiers that voted for the ith class, *i.e.*,

$$y_i = \sum_{j=1}^{m} [\![y_{ij} > 0]\!] \tag{B.72}$$

where y_{ij} is the SVM output of the pairwise classifier between i and j. $y_{ij} > 0$ when the pairwise classifier votes for the ith classifier, otherwise, it votes for the jth classifier, $[\![\cdot]\!]$ indicates

$$[\![X]\!] = \begin{cases} 1 & \text{if } X \text{ true} \\ 0 & \text{if } X \text{ false} \end{cases}$$

$$y_{ij} = \sum_{r=0}^{n} \omega_{ijr} x_r$$

$$\omega_{ijr} = \begin{cases} w_{ijr} & \text{, if } r > 0 \\ -b & \text{, if } r = 0 \end{cases}$$

where w_{ijr} and b are the original learned coefficients and constant from the pairwise classifier for y_{ij}.

We want to express the weights of evidence in terms of input features, $r = 0 \dots n$. We can multiple Equation (B.72) with the normalized sum of weights of evidence due to features:

$$\frac{1}{\text{value}\left(y_{ij}\right)} \sum\nolimits_{r=0}^{n} \omega_{ijr} x_r = 1 \tag{B.73}$$

where value(·) = sgn(·)|·| and we denote the total evidence due to features as $\text{value}\left(y_{ij}\right)$. This represents the number value of y_{ij} rather than y_{ij} as a series expansion. Multiplying Equation (B.72) with Equation (B.73) and rearranging, we get

$$\Delta g_{ij} = \sum\nolimits_{r=0}^{n} \Delta f_{ijr} \tag{B.74}$$

where

$$\Delta f_{ijr} = \frac{\omega_{ijr}}{\text{value}\left(y_{ij}\right)} x_r [\![y_{ij} > 0]\!]$$

$$\omega_{ijr} = \begin{cases} w_{ijr} & \text{, if } r > 0 \\ -b & \text{, if } r = 0 \end{cases}$$

B.14.4 PAIRWISE COUPLING

Before we discuss multiclass calibrated linear SVMs, we discuss *pairwise coupling*, an important method to support multiclass inference with binary classifiers.

Many inference models can only make classifications between two class values (*e.g.*, Logistic Regression, Support Vector Machines), but applications may want to infer over more than two value (*i.e.*, a multi-class problem). Several methods exist to help use binary classifiers (that deal with only two values at a time) for multi-class problems. For example, n one-*vs.*-all classifiers can be used to infer over n class values, and the value of the classifier with the highest certainty is chosen. An alternative popular approach is to use Pairwise Coupling [Hastie and Tibshirani, 1998] that looks at n one-*vs.*-one classifiers, and chooses the class value that is selected by the most classifiers. We briefly introduce this algorithm in this section, and generate explanations for it. Refer to the original paper for a more detailed discussion of the method.

For a multiclass problem with n class values, consider a system of n^2 pairwise classifiers covering all pairwise comparisons. We index these classifiers as $\{ij\}$ representing the binary classifier that will infer the ith class if the inference is positive, and the jth class instead if the inference is negative. For pairwise classifiers that are calibrated, we can denote the probability of inferring the ith class instead of the jth class with r_{ij}. These probabilities can be represented in a matrix:

$$\{r_{ij}\} = \begin{pmatrix} \cdot & r_{12} & \cdots & r_{1n} \\ r_{21} & \cdot & \cdots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n1} & r_{n2} & \cdots & \cdot \end{pmatrix} = \begin{pmatrix} \cdot & r_{12} & \cdots & r_{1n} \\ 1 - r_{12} & \cdot & \cdots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 1 - r_{1n} & 1 - r_{n2} & \cdots & \cdot \end{pmatrix} = \{1 - r_{ji}\} \qquad \text{(B.75)}$$

We wish to estimate the true probabilities for each class value, p_i (estimated as $\hat{p}_i$) from this system of pairwise comparisons. In terms of these true probabilities, the probability outcome of each pairwise classifier can be modeled as:

$$\mu_{ij} = E(r_{ij}) = \frac{p_i}{p_i + p_j} \qquad \text{(B.76)}$$

where r_{ij} is the observed (measured) pairwise classification probability, and μ_{ij} is the model (theoretical) pairwise probability. The Pairwise Coupling algorithm estimates $\hat{p}_i$ by maximizing the negative Kullback-Leibler distance between r_{ij} and the estimates of μ_{ij}, notated as $\hat{\mu}_{ij}$. $\hat{p}_i$ converges iteratively with:

$$\hat{p}_i \leftarrow \hat{p}_i \frac{\sum_{j \neq i} n_{ij} r_{ij}}{\sum_{j \neq i} n_{ij} \hat{\mu}_{ij}} \qquad \text{(B.77)}$$

where n_{ij} is the weight for the $\{ij\}$th pairwise classifier. As shorthand, we drop the hat notation to indicate the estimate:

$$\mu_{ij} = r_{ij} + \ell_{ij} \qquad \text{(B.78)}$$

where $\ell_{ij}(x) = \mu_{ij} - r_{ij}$ is the minimized Kullback-Leibler distance between the estimate μ_{ij} from the observed r_{ij}. ℓ_{ij} is a function of x and its value varies for each inference. We assume ℓ_{ij} is small, but we want to retain this difference, because otherwise approximating it away can lead to fallacious explanations.

Rearranging Equation (B.76), we can derive an expression for p_i in terms of the probability of another class, p_j:

$$p_i = \mu_{ij}(p_i + p_j)$$

$$p_i = \left(\frac{\mu_{ij}}{1 - \mu_{ij}}\right) p_j \tag{B.79}$$

Note that, unlike how the multinomial logistic regression deals with the multiclass problem. we do not have an expression for p_i independent of other classes. We therefore cannot derive a one-*vs.*-none Weights of Evidence explanation, *i.e.*,

$$g_i = \{\emptyset\} \tag{B.80}$$

However, we can derive one-*vs.*-one Weight of Evidence explanations, Δg_{ij}, about pairwise comparisons. From this, we may derive the one-*vs.*-all explanation, $\Delta g_{i\forall}$. To derive an explanation for why the ith class was inferred over the jth we start with the fact

$$p_i \geq p_j$$

$$\left(\frac{\mu_{ij}}{1 - \mu_{ij}}\right) p_j \geq p_j$$

$$\frac{\mu_{ij}}{1 - \mu_{ij}} \geq 1$$

Taking a log transforms gives us a logit expression in terms of μ_{ij}:

$$\log\left(\frac{\mu_{ij}}{1 - \mu_{ij}}\right) = \text{logit}(\mu_{ij}) \geq 0 \tag{B.81}$$

At this point, we can express a one-*vs.*-one Weights of Evidence explanation for the inference of ith class over the jth class in terms of estimated pairwise classifier probability μ_{ij}:

$$\Delta g_{ij} = \sum\nolimits_{j=1}^{m} \text{logit}(\mu_{ij}) \tag{B.82}$$

where μ_{ij} suggests some weight of evidence for $p_i > p_j$.

Note that this method is agnostic to the base pairwise classifier, so even a "black box" classifier can be used.

$$\Delta g_{ij} \cong \sum_{j=1}^{m} \mu_{ij} \tag{B.83}$$

While we can generate explanations for the Pairwise Coupling step, we can leave the pairwise classifier as unexplained, or explained as a *separate layer*. Fortunately, as long as the logit expression $\text{logit}(r_{ij})$ can be expanded, we can provide closed explanation in terms of input features. This is possible for multiclass SVMs, which we discuss next.

B.14.5 MULTICLASS PAIRWISE-COUPLED LINEAR SVM

Linear SVMs are often used for multi-class problems (*i.e.*, where inferences need to be made over ≥ 2 class values). To handle multiple class values, pairwise coupling [Hastie and Tibshirani, 1998] can be used on all the pairwise classifiers. In fact, this is what the Weka toolkit uses for multi-class SVMs. We have already described how to generate explanations of pairwise coupling of classifiers in Section B.14.4. For the multiclass case, we express Equation (B.70) for each pairwise classifier, with a change in notation $p = r_{ij}$ and other additions of the $\{ij\}$ subscripts:

$$r_{ij} = \frac{1}{1 + e^{-(\beta_{ij} \sum_{r=0}^{n} w_{ijr} x_r + \gamma_{ij})}} \tag{B.84}$$

Similarly to Equation (B.71), the logit of Equation (B.84) is a linear expression:

$$\text{logit}(r_{ij}) = \beta_{ij} \sum_{r=0}^{n} w_{ijr} x_r + \gamma_{ij} \tag{B.85}$$

Equation (B.85) only provides the weights of evidence explanation for a single pairwise classifier. In order to explain the overall inference, we seek to expand on Equation (B.82), noting that while $\mu_{ij} \approx r_{ij}$, they are not equal ($\mu_{ij} \neq r_{ij}$), and

$$\begin{aligned} \text{logit}(\mu_{ij}) &\approx \text{logit}(r_{ij}) \\ &\neq \text{logit}(r_{ij}) \end{aligned} \tag{B.86}$$

We can address this discrepancy by treating it as a difference error or a scale error.

B.14.5.1 DISCREPANCY BY DIFFERENCE

By considering the discrepancy as a difference, we have:

$$\begin{aligned} \text{logit}(\mu_{ij}) &= \text{logit}(r_{ij}) + \delta_{ij} \\ &= \left(\sum_{r=0}^{n} \beta_{ij} w_{ijr} x_r + \gamma_{ij}\right) + \delta_{ij} \end{aligned} \tag{B.87}$$

Therefore, from Equation (B.82), we can produce the explanation:

$$\Delta g_{ij} = \text{logit}(\mu_{ij}) = \sum_{r=0}^{n} \Delta f_{ijr} \tag{B.88}$$

where

$$\Delta f_{ijr} = \omega_{ijr} x_r$$

$$\omega_{ijr} = \begin{cases} \beta_{ij} w_{ijr} & \text{, if } r > 0 \\ \beta_{ij} w_{ij0} + \gamma_{ij} + \delta_{ij} & \text{, if } r = 0 \end{cases}$$

$$\delta_{ij} = \text{logit}(\mu_{ij}) - \text{logit}(r_{ij})$$

$$x_0 = 1$$

r_{ij} is output from each pairwise classifier, and μ_{ij} is estimated from the pairwise coupling method. For each inference, the values of r_{ij} and μ_{ij} are computed, so we can just use them as known values, rather than functions in x. Notice that ω_{ir} is independent of ℓ_{ij} for $r > 0$, but ω_{i0} is dependent on the test instance, x, used for inference. Thus, we can call ω_{ij0} the evidence due to estimation.

Equation (B.88) reduces well to the binary class case:

$$\Delta g_{ij} = \sum_{r=0}^{n} \Delta f_{ijr} = \sum_{r=0}^{n} \Delta f_r$$

where $\Delta f_r = \omega_r x_r$ and $\ell_{ij} = 0$, so $\omega_r = \beta w_r + \gamma$.

This difference treatment may suffer from cases where the value of δ_{ij} may be very large in magnitude. We can ameliorate this issue by considering the discrepancy as due to scaling.

B.14.5.2 DISCREPANCY BY SCALING

By considering the discrepancy as due to scaling, we have:

$$\begin{aligned} \text{logit}(\mu_{ij}) &= \kappa_{ij}\text{logit}(r_{ij}) \\ &= \sum_{r=0}^{n} \kappa_{ij}\beta_{ij}w_{ijr}x_r + \kappa_{ij}\gamma_{ij} \end{aligned} \tag{B.89}$$

Therefore, from Equation (B.82), we can produce the explanation:

$$\Delta g_{ij} = \text{logit}(\mu_{ij}) = \sum_{r=0}^{n} \Delta f_{ijr} \tag{B.90}$$

where

$$\Delta f_{ijr} = \omega_{ijr}x_r$$

$$\omega_{ijr} = \begin{cases} \kappa_{ij}\beta_{ij}w_{ijr} & \text{, if } r > 0 \\ \kappa_{ij}\beta_{ij}w_{ij0} + \kappa_{\text{ij}}\gamma_{ij} & \text{, if } r = 0 \end{cases}$$

$$\kappa_{ij} = \frac{\text{logit}(\mu_{ij})}{\text{logit}(r_{ij})}$$

B.15 NAÏVE BAYES EXPLAINER

The naïve Bayes classifier is a simple classifier that uses Bayes theorem to classify values. It assumes that input features are conditionally independent of one another. Our explanation for naïve Bayes inference is an extension of [Poulin et al. 2006] for multi-class problems. The posterior probability that the ith class is inferred $(y = y_i)$ from a set of m class values given the observed instance input feature values x:

$$P(y = y_i|x) = P(y_i|x_1, x_2, \cdots, x_n) \cong P(y_i) \prod_{r=1}^{n} P(x_r|y_i) \tag{B.91}$$

where x_r is an input feature of n possible values. We neglect the denominator term as it will cancel out. The probability is calculated from the prior probability that a class would be in, $P(y_i)$, and the conditional probabilities of each feature value given the class, $P(x_r|y_i)$.

For notation convenience, let us define

$$p_{ir} = \begin{cases} P(y_i) & r = 0 \\ P(x_r|y_i) & r > 0 \end{cases}$$

$$p_i = P(y_i|x)$$

Then Equation (B.91) becomes

$$p_i \cong \prod_{r=0}^{n} p_{ir} \tag{B.92}$$

Taking a log transform of Equation (B.92) gives the linear expression for the weights of evidence explanation:

$$\begin{gathered} \log(p_i) = \sum_{r=0}^{n} \log(p_{ir}) \\ g_i = \sum_{r=0}^{n} f_{ir} \end{gathered} \tag{B.93}$$

where $f_r = \log(p_{ir})$.

B.16 HIDDEN MARKOV MODELS (HMM) EXPLAINER

A hidden Markov model (HMM) is a Bayesian network that models the probability of a *sequence* of hidden states given a sequence of observations (input features with respect to time). First-order Markov models assume that only the previous state affects the next, and only the current state influences the current observation. For detailed information on HMMs, please refer to [Rabiner, 1989]. We can derive a weights of evidence explanation for HMMs in a similar manner as we did for naïve Bayes.

B.16.1 Single Observation per Time Step

We consider a sequence of length T. The probability of inferring state (Output) sequence s, given the observation (Inputs) sequence x is:

$$\begin{aligned} P(s|x) &= P(s_1)\left(\prod_{t=2}^{T} P(s_t|s_{t-1})\right)\left(\prod_{t=1}^{T} P(x_t|s_t)\right) \\ &= \pi(s_1)\left(\prod_{t=2}^{T} A(s_t|s_{t-1})\right)\left(\prod_{t=1}^{T} B(x_t|s_t)\right) \end{aligned} \tag{B.94}$$

where $P(s_1) = \pi(s_1)$ is the *prior* probability that a state is s_1, $P(s_t|s_{t-1}) = A(s_t|s_{t-1})$ the *transition* probabilities from state s_{t-1} to s_t, and $P(x_t|s_t) = B(x_t|s_t)$ the *emission* probabilities of the observations given the state sequence.

Taking a log transform on Equation (B.94) gives the weights of evidence explanation:

$$\begin{aligned} g_s &= \log\big(P(s|x)\big) \\ &= \sum_{t=1}^{1} \log\big(\pi(s_1)\big) + \sum_{t=2}^{T} \log\big(A(s_t|s_{t-1})\big) + \sum_{t=1}^{T} \log\big(B(x_t|s_t)\big) \\ &= \sum_{t=1}^{T} f_{st} \end{aligned} \tag{B.95}$$

where

$$f_{st} = \begin{cases} \log\big(\pi(s_1)\big) + \log\big(B(x_t|s_t)\big) & \text{, if } t = 1 \\ \log\big(A(s_t|s_{t-1})\big) + \log\big(B(x_t|s_t)\big) & \text{, if } t > 1 \end{cases}$$

B.16.2 Observation Vector per Time Step

Next, we consider HMMs with observation vectors for each time step, specifically with n input features. At time step t, the observation is $\vec{x}_t = (x_{t1}, x_{t2}, \dots, x_{tn})$. The probability of inferring state sequence s, given the observation sequence x is:

$$\begin{aligned} P(s|x) &= P(s_1)\left(\prod_{t=2}^{T} P(s_t|s_{t-1})\right)\left(\prod_{t=1}^{T} P(\vec{x}_t|s_t)\right) \\ &= \pi(s_1)\left(\prod_{t=2}^{T} A(s_t|s_{t-1})\right)\left(\prod_{t=1}^{T} B(\vec{x}_t|s_t)\right) \end{aligned} \tag{B.96}$$

To allow features to individually provide evidence, we need to make a *naive assumption* (similar to what is done for naive Bayes) that the features are independent of one another given any state, *i.e.*,

$$P(\vec{x}_t|s_t) = P\left(x_{t1}, x_{t2}, \dots, x_{tn}\middle|x_t^i\right) \cong \prod\nolimits_{r=1}^{n} P(x_{tr}|s_t) \tag{B.97}$$

We define a new parameter for the HMM, $\tilde{B}(x_{tr}|s_t) = P(x_{tr}|s_t)$, which is a naive emissions probability matrix representing the probability of observing input value x_{tr} given hidden state s_t. This can be computed with a labeled training set.

Substituting Equation (B.97) into Equation (B.96) gives

$$P(s|x) = \pi(s_1)\left(\prod\nolimits_{t=2}^{T} A(s_t|s_{t-1})\right)\left(\prod\nolimits_{r=1}^{n} \tilde{B}(x_{tr}|s_t)\right) \tag{B.98}$$

For notation convenience, we rewrite Equation (B.98) as:

$$p_s \cong \prod\nolimits_{t=1}^{T}\prod\nolimits_{r=0}^{n} p_{str} \tag{B.99}$$

where

$$p_{str} = \begin{cases} \pi(s_1) & \text{, if } r = 0, t = 1 \\ A(s_t|s_{t-1}) & \text{, if } r = 0, t > 1 \\ \tilde{B}(x_{tr}|s_t) & \text{, if } r > 0 \end{cases}$$

Taking a log transform on Equation (B.96) gives the weights of evidence explanation:

$$g_s = \sum\nolimits_{t=1}^{T}\sum\nolimits_{r=0}^{n} f_{str} \tag{B.100}$$

where

$$f_{str} = \log(p_{str}) = \begin{cases} \log\left(\pi(s_1)\right) & \text{, if } r = 0, t = 1 \\ \log\left(A(s_t|s_{t-1})\right) & \text{, if } r = 0, t > 1 \\ \log\left(\tilde{B}(x_{tr}|s_t)\right) & \text{, if } r > 0 \end{cases}$$

So, with the naive assumption of independence among features, HMMs can be explained as the sum of evidence of:

1. Prior probabilities of **selected state**, $(r = 0, t = 1)$

2. Weights of Evidence due to each **state transition**, $(= 0, t > 1)$
3. Weights of Evidence due to **feature value** at **sequence step**, $(r > 0)$

B.16.3 EXPLANATIONS REGARDING THE LAST TIME STEP

Even though the HMM will infer over a time sequence, end-users may be interested only in the last time step. In this case, explanations will pertain to the various n class values that the last state s_T may take. Given the inferred state sequence, $s = (s_1, s_2, \dots, s_t, \dots, s_T)$, we consider the last state inferred as the ith class value, *i.e.*, $s_T = i$. We want to present the evidence for $s_T = i$, $i \in [1, m]$. given $s_1, s_2, \dots, s_{T-1}$; this explains the probability of inference, $P(s_T = i | s_1, s_2, \dots, s_{T-1})$. To use the same notation as non-time-based explainers, we also define $x_r = x_{Tr}$. Fixing $s_1, s_2, \dots, s_{T-1}$, Equation (B.100) becomes:

$$g_i = \sum_{t=1}^{T} \sum_{r=0}^{n} f_{itr} \tag{B.101}$$

where

$$f_{itr} = \begin{cases} \log\big(\pi(s_1)\big) & \text{, if } r = 0, t = 1 \\ \log\big(A(s_t | s_{t-1})\big) & \text{, if } r = 0, 1 < t < T \\ \log\big(A(i | s_{T-1})\big) & \text{, if } r = 0, t = T \\ \log\left(\tilde{B}(x_{tr} | s_t)\right) & \text{, if } r > 0, 1 < t < T \\ \log\left(\tilde{B}(x_r | i)\right) & \text{, if } r > 0, t = T \end{cases}$$

Equation (B.101) presents the weights of evidence in two dimensions, time step and input features.

We can now use the techniques for non-time-based explainers to derive explanations for Why, Why Not questions. However, each weights of evidence, g_i, explains the conditional probability $p_i = P(s_T = i | s_1, s_2, \dots, s_{T-1})$ instead of $P(s_T = i)$.

B.17 DECISION STUMP EXPLAINER

A decision stump is one-level decision trees (*i.e.*, the root and leaves) [Iba and Langley, 1992], which uses a single input feature for inference. Decision stumps are typically used as part of an ensemble classifier, such as AdaBoost. It infers a probability distribution over the class values.

Denoting the selected feature as the $\tilde{r}$th feature and the probability for inferring the ith class value as p_i, the weights of evidence explanation is:

$$g_i = \sum\nolimits_{r=0}^{n} f_{ir} \tag{B.102}$$

where $f_{ir} = p_i[\![r = \tilde{r}]\!]$, and $[\![\cdot]\!]$ is the Kronecker delta notation.

This explainer algorithm is implemented in `DecisionStumpExplainer`.

B.18 Decision Tree Evidence Explainer

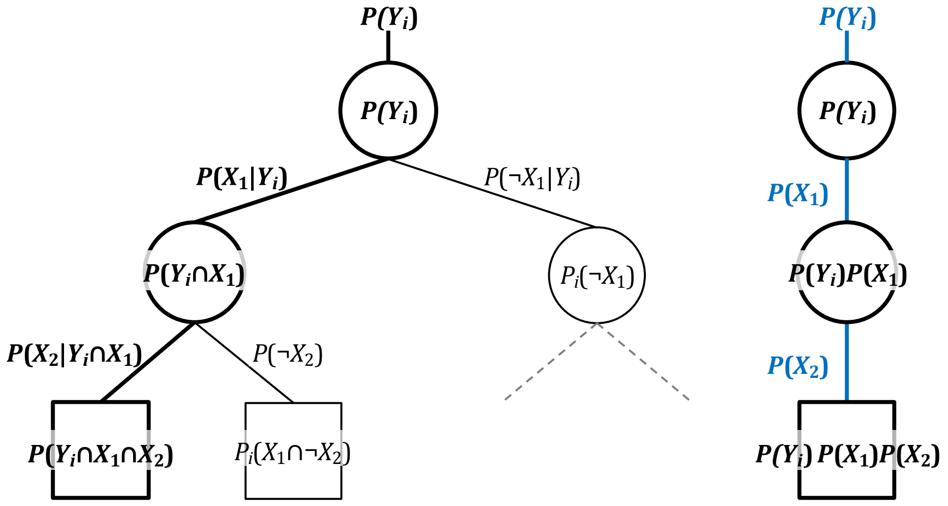


Figure B.7. (Left) A decision tree illustrating probabilities at each node due to the probability distribution in the training set satisfying the feature conditions, X_ρ, at the node, and conditional probabilities of each edge between nodes. (Right) Probabilities of nodes and edges of the reasoning trace, assuming conditional independence between feature conditions. Edge probabilities are used in the weights of evidence explanation.

We had previously considered the decision tree deterministically as a combination of rules. Here, we want to consider it probabilistically, where its structure is due to statistical treatment of a training set. This will allow us to produce a Weights of Evidence explanation *of the trace*, where we attribute atomic weights of evidence to each decision condition in the reasoning trace. Given a trace with η conditions, we denote the ρth condition as X_ρ. A condition may be an equality or inequality, *e.g.*, $x_r = 10$, $x_r > 7$, where x_r refers to the rth input feature.

A full trace is represented by the event

$$X_1 \cap X_2 \cap \dots \cap X_\eta = \bigcap_{\rho=1}^{\eta} X_\rho$$

A learned decision tree encodes the probability distribution for inferring class values as the probability distribution of the training instances at the leaf of the traced path in the tree. So the probability of inferring the ith class (event denoted as $Y_i \colon y = i$) is

$$p_i = P_i\left(\bigcap_{\rho=1}^{\eta} X_\rho\right) = P\left(Y_i \cap \bigcap_{\rho=1}^{\eta} X_\rho\right) \tag{B.103}$$

We will use the LHS notation for the rest of the working. We can similarly recover the probability of inferring the ith class for a pruned tree (*i.e.*, a truncated reasoning trace). Consider the reasoning trace truncated to the ρth node (condition). The probability of inferring the ith class can be computed as

$$P_i\left(\bigcap_{q=1}^{\rho} X_q\right) = \frac{n_i\left(\bigcap_{q=1}^{\rho} X_q\right)}{n_i\left(\bigcap_{q=1}^{\rho-1} X_q\right)} \tag{B.104}$$

where $n_i\left(\bigcap_{q=1}^{\rho} X_q\right)$ is the number of training instances satisfying the condition $\bigcap_{q=1}^{\rho} X_q$ and which are labeled with the ith class.

Note that this does not represent the atomic weight of evidence due to the condition X_r. Instead, the atomic weight is the probability of transition along the edge from X_{r-1} to X_r:

$$P_i\left(X_\rho \middle| Y_i \cap \bigcap_{q=1}^{\rho-1} X_q\right) = \frac{P_i\left(\bigcap_{q=1}^{\rho} X_q\right)}{P_i\left(\bigcap_{q=1}^{\rho-1} X_q\right)} \tag{B.105}$$

Unfortunately, in this edge probability depends on X_r, due to conditional dependence. However, if we take assume conditional independence between $X_r, \forall r$, as is the case for naïve Bayes, then

$$P_i\left(X_\rho \middle| Y_i \cap \bigcap_{\rho=1}^{r-1} X_\rho\right) \cong P_i(X_\rho) = p_{i\rho} \tag{B.106}$$

Substituting Equation (B.119) into Equation (B.116) gives

$$p_i \cong p_{i0} p_{i1} p_{i2} \dots p_{i\eta} = \prod_{\rho=0}^{\eta} p_{i\rho} \tag{B.107}$$

where $p_{i0} = P(Y_i) = P(y = i)$ is the prior probability for inferring the ith class. p_{i0} can be calculated from the distribution of the full training set.

Taking a log transform of Equation (B.120) gives the Weights of Evidence explanations of a reasoning trace inferring the ith class:

$$g_i = \sum_{\rho=1}^{\eta} f_{i\rho} \tag{B.108}$$

where η is the number of conditions in the trace, ρ refers to the condition index in the trace,

$$f_{i\rho} \cong \log(p_{i\rho})$$

$$p_{i\rho} = \begin{cases} P(Y_i) & \text{, if } r = 0 \\ P\left(Y_i \cap \bigcap_{q=1}^{\rho} X_q\right) \Big/ P\left(Y_i \cap \bigcap_{q=1}^{\rho-1} X_q\right) & \text{, if } r > 0 \end{cases}$$

$$Y_i\colon \quad y = y_i \quad \equiv \quad i\text{th class inferred}$$

With this derivation, we will be able to provide alternative explanations in terms of Weights of Evidence, instead of Reasoning Traces. This can be particularly insightful for the Why explanation by identifying factors that are more influential. However, it may not be as meaningful for Why Not and How To explanations, since it provides less actionable information than reasoning traces.

This method produces a different explainer than the deterministic decision tree one. It generates a Weights of Evidence instead of a Rule Trace explanation, and has corresponding explanations for Why, Why Not and How To, for the specific rule trace that was traversed. Note that while the Rule Trace explanation can cover multiple traces, the Weights of Evidence explanation only handles one. Figure B.8and Figure B.9 describe the algorithm for preparing the probability distribution in Equation (B.107).

```
$\tau_0$ = root node of decision tree
$X_0$ := condition literal representing edge $\{\emptyset, \tau_0\}$ from null to root node
$\xi$ := new empty trace clause
$\Psi = \bigcup_{j=1}^{m} \psi_j$ = map of DNFs of each class value // gets updated in TreetoDNF
TreeToDNF($\tau_0, X_0, \emptyset, \xi, \Psi$)

TreeToDNF($\tau, X, P_{\tau-1}, \xi, \Psi$)
    // compute edge probabilities from node probabilities; see Equation (B.105)
    $P_{\tau-1} = \sum_{j=1}^{m} P_{\tau-1,j}$ = probability class distribution of parent node $\tau - 1$
    $P_\tau = \sum_{j=1}^{m} P_{\tau,j}$ := read probability class distribution at current node $\tau$
    $P(X) = \sum_{j=1}^{m} p_i$ := EdgeProbabilities($P_{\tau-1}, P_\tau$)
    Store $P(X)$ into $X$

    If $\tau$ parent of $\sum_v \tau_v$ for some $\tau_v$ // $\tau$ has child nodes
        $x_r$ = attribute described in $\tau$
        For each child $\tau_v$
            $X_v$ := EdgeLiteral($\tau, \tau_v$) // define condition literal to represent edge $\{\tau, \tau_v\}$ from $\tau$ to $\tau_v$
            $\xi_v$ := Clone $\xi$ // duplicate trace to extend for each child path
            $\xi_v := \xi_v + X_v$ // append literal $X_v$ to trace clause $\xi_v$
            TreeToDNF($\tau_v, X_v, P_\tau, \xi_v, \Psi$)
    Else // $\tau$ is a leaf
        $i := \max_i P_\tau$ // inferred class value
        $\psi_i = \Psi \cdot [\![ j = i ]\!]$ // retrieve DNF $\psi_i$ that infers the $i$th class
        $\psi_i := \psi_i + \xi$ // append trace clause $\xi$ to $\psi_i$
```

Figure B.8. Modified algorithm (of Figure B.5) to traverse the decision tree depth-first to convert it into DNF and store class probability distributions of edge conditions.

```
EdgeProbabilities ($P_{\tau-1}, P_\tau$)
    $P = \sum_{j=1}^{m} p_i$
    For each class value index $j$
        $p_i := P_{\tau,j} / P_{\tau-1,j}$ // implements Equation (B.105)
    Return P
```

Figure B.9. Algorithm that implements Equation (B.105) to calculate the class probability probability at each edge transition in the decision tree.

This explainer algorithm is implemented in `J48EvidenceExplainer`. Furthermore, to support weights of evidence explanations for the ensemble classifier RandomForest [Breiman, 2001], this explainer algorithm is also implemented in `RandomTreeExplainer` to explain Random Trees.

B.18.1 WEIGHTS OF EVIDENCE OF INPUT FEATURES INSTEAD OF CONDITIONS

Equation (B.108) presents the weights of evidence explanation in terms of feature conditions of the reasoning trace. This differs from other weights of evidence explainers (*e.g.*, for linear SVM, logistic regression, and naïve Bayes) because some input features may be omitted or may occur in multiple feature conditions in the trace. Fortunately, we can still provide an explanation in terms of input features by summing weights of evidence of each condition, $f_{i\rho} = \log(p_{i\rho})$, that has the same input feature, x_r:

$$g_i = \sum_{r=0}^{n} f_{ir} \tag{B.109}$$

where

$$f_{ir} = \begin{cases} \log(p_{i0}) & \text{, if } r = 0 \\ \sum_{\rho=0}^{\eta} \log\left(\frac{p_{i\rho}}{\epsilon}\right) [\![x_r \in X_\eta]\!] & \text{, if } r > 0 \end{cases}$$

$$[\![x_r \in X_\eta]\!] = \begin{cases} 1 & \text{, if } X_\eta \text{ describes } x_r \\ 0 & \text{, otherwise} \end{cases}$$

This formulation of the weights of evidence explanation in terms of input features will also be useful when combining multiple explanations to explain an ensemble classification (see Section B.20).

B.19 K-NEAREST NEIGHBORS (KNN) EXPLAINER

The k-Nearest Neighbors algorithm (kNN) is an instance-based learning method that classifies outcomes based on the instances in the training set that are "nearest" to the test instance, x_0. These training set instances are called neighbors, and we denote the kth nearest neighbor as x_k. The distance between instances depend on the differences between each input feature value,

$$\Delta x_{kr} = |x_{0r} - x_{kr}|$$

where x_{0r} is the value of the rth input feature for the test instance, and x_{kr} is the value of the rth input feature for kth neighbor.

The total distance over n input features is calculated with a Distance function, δ, most typically Euclidean or Manhattan,

$$\delta_k = \delta(x_0, x_k) = \begin{cases} \sqrt{\sum_{r=1}^{n} \Delta x_{kr}^2} & \text{Euclidean} \\ \sum_{r=1}^{n} \Delta x_{kr} & \text{Manhattan} \end{cases}$$

Up to k nearest neighbors that are selected will vote to assign the class value to the test instance that matches the majority class value among the nearest neighbors. This depends on the Weighting function,

$$w_k = w(\delta_k) = \begin{cases} 1 & \text{None} \\ 1 - \delta_k & \text{Similarity} \\ 1/\delta_k & \text{Inverse} \end{cases}$$

Therefore, the probability for inferring the ith class is

$$p_i = \frac{1}{Z}\sum_{k=1}^{J} w_k = [\![y_k = i]\!] \tag{B.110}$$

where Z is a normalization constant, J is the number of neighbors being considered for classification, $w_k = w\big(\delta(x_0, x_k)\big)$, and we use the Kronecker delta notation:

$$[\![y_k = i]\!] = \begin{cases} 1 & \text{If the } k\text{th neighbor is labeled with the } i\text{th class} \\ 0 & \text{Otherwise} \end{cases}$$

We can also represent the inference as

$$p_i \propto \sum\nolimits_{k \in K_i} w_k \tag{B.111}$$

where K_i is the the set of neighbors labeled with the ith class.

We would like to produce an explanation in terms of two dimensions: input features ($r = 1 \dots n$), and nearest neighbors ($k = 1 \dots J$). We will first derive an explanation for several Distance functions, and then for the Weighting functions, combining them into a single explanation expression, rather than separated layers.

B.19.1 DISTANCE FUNCTIONS

To produce a Weights of Evidence explanation in terms of input features, we need to derive a linear additive expression for the distances.

B.19.1.1 MANHATTAN DISTANCE

The Manhattan distance (also called Taxi-Cab distance) is a simplest distance function that takes the L1-norm:

$$\delta_k = \sum\nolimits_{r=1}^{n} \delta_{kr} \tag{B.112}$$

where $\delta_{kr} = \Delta x_{kr}$ is the distance for the rth input feature. This is already a linear additive expression of feature differences.

B.19.1.2 EUCLIDEAN DISTANCE

The Euclidean distance is a common distance function used in kNN and it is also default in Weka. It is an L2-norm non-linear expression in terms of input feature distances, $\delta_k = \left(\sum_{r=1}^{n} \Delta x_{kr}^2\right)^{1/2}$.

To be able to express the Euclidean distance, δ_k, in terms of Weights of Evidence due to the distance of each input feature, δ_{kr}, we seek a linear approximation for the Euclidean distance. In the field of computer graphics and engineering applications, Celebi, Celikerb, and Kingravi [2011] compared several linear approximation methods for the Euclidean distance of arbitrary dimensions, and found that although Barni *et al.*'s [1995; 2000] approximation was the most computationally inefficient, it was the most accurate with the lowest maximum relative error (MRE). Since we are primarily interested in an accurate explanation than speed of computing it, we choose this representation. For an Euclidean distance,

$$\delta = |\vec{\delta}| = \sqrt{\sum\nolimits_{r=1}^{n} \delta_r^2}$$

where δ_r represents the rth orthogonal component distance, and $\vec{\delta}$ represents the distance vector, the approximation, D_B, takes the linear form:

$$\delta \approx D_B(\vec{\delta}) = \gamma \sum\nolimits_{r=1}^{n} \alpha_r \delta_{(r)} \tag{B.113}$$

where $\delta_{(r)}$ is a permutation of $\{|\delta_r|\} = \langle |\delta_1|, |\delta_2|, \ldots, |\delta_n| \rangle$ such that $\delta_{(1)} \geq \delta_{(2)} \geq \cdots \geq \delta_{(n)}$ (henceforth called value-ranked), and α_r and $\gamma > 0$ are approximation parameters, estimated with:

$$\hat{\alpha}_r = \sqrt{r} - \sqrt{r-1}$$

$$\hat{\gamma} = \frac{2}{1 + \sqrt{\sum_{r=1}^{n} \hat{\alpha}_r^2}}$$

These parameters are constant only depend on the number of input features, n. The maximum relative error (MRE) for this approximation is $\varepsilon_{\max}^{D_B} = 1 - \hat{\gamma}$.

Hence, we can provide the explanation for Euclidean distance as linear approximation:

$$\delta_k \approx \sum\nolimits_{r=1}^{n} \delta_{kr} \tag{B.114}$$

where $\delta_{kr} = \lambda_r \Delta x_{k,(r)}$ and $\lambda_r = \hat{\gamma}\hat{\alpha}_r$.

B.19.1.3 FEATURE DISTANCES FOR NUMERIC AND CATEGORICAL FEATURES

Depending on the type of the input feature (numeric, normalized numeric, or categorical), distance due to each input feature will differ:

$$\Delta x_{kr} = \begin{cases} x_{0r} - x_{kr} & \text{numeric} \\ \text{norm}(x_{0r}) - \text{norm}(x_{kr}) & \text{numeric, normalized} \\ 0 & \text{categorical}, x_r = x_{kr} \\ 1 & \text{otherwise} \end{cases} \tag{B.115}$$

For normalization, Weka uses min-max normalization for its kNN implementation:

$$\text{norm}(x_r) = \frac{x_r - x_{\text{r,min}}}{x_{\text{max}} - x_{\text{r,min}}} \tag{B.116}$$

Given the aforementioned linear expression for distance functions, we next turn our attention to finding linear expressions for weighting functions.

B.19.2 WEIGHTING FUNCTIONS

We seek to derive linear expressions for in terms of distance and consequently get linear expressions for weights of evidence in terms of feature distances:

$$\begin{aligned} g_i &= \sum\nolimits_{k \in K_i} w_k \\ &= \sum\nolimits_{k \in K_i} \sum\nolimits_{r=1}^{n} \omega_{kr} \end{aligned} \tag{B.117}$$

where $w_k = \sum_{r=1}^{n} \omega_{kr}$ is the weighting function in terms of the distance δ_k and ω_{kr} represents the atomic weight of evidence. We derive explanations for three weighting functions (None, Similarity, and Inverse).

B.19.2.1 NO WEIGHTING (NONE)

No weighting is done for this function, such that $w_k = 1, \forall k$. So we get the weights of evidence:

$$\begin{aligned} g_i &= \sum\nolimits_{k \in K_i} 1 = |K_i| \\ &= \sum\nolimits_{k \in K_i} \sum\nolimits_{r=1}^{n} \frac{1}{n} \end{aligned} \tag{B.118}$$

This weighting function has no concept of explanation by input features.

B.19.2.2 SIMILARITY WEIGHTING

Similarity weights each farther nearest neighbor less with the function, $w_k = 1 - \delta_k$. So we get the weights of evidence:

$$\begin{aligned} g_i &= \sum\nolimits_{k \in K_i} (1 - \delta_k) \\ &= \sum\nolimits_{k \in K_i} \sum\nolimits_{r=1}^{n} \omega_{kr} \end{aligned} \tag{B.119}$$

where $\omega_{kr} = \frac{1}{n} - \delta_{kr}$.

B.19.2.3 INVERSE WEIGHTING

Similarity weights each farther nearest neighbor less with the function, $w_k = 1/\delta_k$. So we get a non-linear expression for the weights of evidence:

$$\begin{aligned} g_i &= \sum\nolimits_{k \in K_i} \frac{1}{\delta_k} \\ &= \sum\nolimits_{k \in K_i} \frac{1}{\sum_{r=1}^{n} \delta_{kr}} \end{aligned} \tag{B.120}$$

where $w_k = \frac{1}{\sum_{r=1}^{n} \delta_{kr}}$.

To get the linear expression of Equation (B.117), we can consider a multivariate Taylor series expansion to the first order. We neglect higher order terms to avoid cross products of terms and maintain linear separability. We expand about the point $\delta_{kr} = \varepsilon$ for $\forall r$, and for some ε, where $0 < \varepsilon \ll 1$. A multivariate Taylor series expansion for w_k gives the expression:

$$\begin{aligned} w_k = \frac{1}{\sum_{r=1}^{n} \delta_{kr}} &\approx \left(w_k |_{\delta_k = \vec{\varepsilon}} \right) + \sum\nolimits_{r=1}^{n} \left(\frac{\partial w_k}{\partial \delta_{kr}} \bigg|_{\delta_k = \vec{\varepsilon}} \right) (\delta_{kr} - \varepsilon) \\ &\approx \frac{1}{n\varepsilon} - \sum\nolimits_{r=1}^{n} \frac{\delta_{kr}}{(n\varepsilon)^2} \\ &= \frac{1}{n\varepsilon} - \frac{1}{(n\varepsilon)^2} \sum\nolimits_{r=1}^{n} \delta_{kr} \end{aligned} \tag{B.121}$$

This expression is similar to weighting by similarity (Equation (B.119)), which we can replicate with a simple scaling and translation, since n and ε are constants:

$$\omega_{kr} = \frac{1}{n} - n\varepsilon + (n\varepsilon)^2 \delta_{kr} = \sum\nolimits_{r=1}^{n} \left(\frac{1}{n} - \delta_{kr} \right)$$

Therefore, under approximation, we can use the same expression as a *surrogate* for explaining inverse weighting as for similarity weighting.

B.19.3 Weights of Evidence for kNN

In summary, we can present the Weights of Evidence explanation as the linear additive expression:

$$g_i = \sum\nolimits_{k=1}^{J} \sum\nolimits_{r=1}^{n} f_{ikr} \tag{B.122}$$

where

$$f_{ikr} = \omega_{ikr} [\![y_k = i]\!]$$

$$\omega_{ikr} = \begin{cases} \dfrac{1}{n} & \text{No Weighting} \\ \dfrac{1}{n} - \delta_{kr} & \text{Similarity Weighting, or} \\ & \text{Inverse Weighting}^{\dagger} \end{cases} \qquad \delta_{kr} = \begin{cases} \Delta x_{kr} & \text{Manhattan} \\ \hat{\gamma}\hat{\alpha}_r \Delta x_{k,(r)} & \text{Euclidean}^{\dagger} \end{cases}$$

$$\hat{\alpha}_r = \sqrt{r} - \sqrt{r-1}$$

$$\hat{\gamma} = \frac{2}{1 + \sqrt{\sum_{r=1}^{n} \hat{\alpha}_r^2}}$$

$$\Delta x_{kr} = |x_{ir} - x_{kr}|$$

† indicates approximations.

B.19.4 How To Explanations – not possible

Since the classifier is built lazily by considering the test instance, it is not possible to provide this explanation without a test instance from which to find neighbors. Furthermore, the explanation expressions are not linearly expressed in terms of input feature values.

B.20 Ensemble Classifier Explainers

To improve the classification accuracy of a single classifier, ensemble methods can be used where multiple classifiers (henceforth called base classifiers) are combined to infer the class. We consider the general case where C base classifiers are used in an ensemble for inference. Many ensemble classifiers make inferences through a linear combination of base classifications, *i.e.*, the probability of inferring the ith class is

$$p_i = \frac{1}{Z}\sum\nolimits_{c=1}^{C} \lambda_c p_{ic} \tag{B.123}$$

where p_{ic} is the probability of the cth classifier inferring value c_i, and Z is a normalization constant.

We wish to derive a weights of evidence expression in terms of weights of evidence of the base explanation:

$$g_i = \sum\nolimits_{c=1}^{C} \lambda_c g_{ic} \tag{B.124}$$

where g_{ic} is the linearly separable weights of evidence expression of lower-level factors (*e.g.*, input features) and λ_c is a coefficient factor for the cth base classifier.

B.20.1 NORMALIZATION OF WEIGHTS OF EVIDENCE OF BASE EXPLANATION

Rather than propagating the probability of inference from a base classifier, some ensemble classifiers discretize classification results of the base classifier to 0 or 1. This discretization neglects much information from the inference in the base classifier …

For base inferences that do influence the ensemble classification, we wish to present their weights of evidence in a form normalized to 1. We do this with the expression:

$$\frac{g_{ic}}{\text{value}(g_{ic})} = 1 \tag{B.125}$$

where g_{ic} is the linearly separable weights of evidence expression for the cth base classifier and $\text{value}(g_{ic}) = \text{sgn}(g_{ic})|g_{ic}|$ is the value of the total weights of evidence.

We can then explain the weights of evidence of the ensemble classification as:

$$g_i = \sum\nolimits_{c=1}^{C} \lambda_{ic} \frac{g_{ic}}{\text{value}(g_{ic})} \tag{B.126}$$

where λ_{ic} is a coefficient factor for the cth base classifier, which depends on the actual ensemble classifier (see Section B.21 and B.22).

B.20.2 TRANSFORMATION-INDEPENDENT NORMALIZATION

If the base explainer formulates g_{ic} as a linear function of p_{ic}, *i.e.*, $g_{ic} = Ap_{ic} + B$ and p_{ic} can be formulated as a linear function of lower-level dimensions (*e.g.*, input features), then g_i can natively present the Weights of Evidence in terms of those dimensions. This is the case for explanations of Decision Stumps, Logistic Regression, binary linear SVM, and kNN.

However, some base classifiers require a transformation of p_{ic} to obtain g_{ic} as a linear additive expression (*e.g.*, explanations for naïve Bayes, HMM, and multiclass SVM require a log transform), *i.e.*,

$$g_{ic} = \varphi(p_{ic})$$

where φ is a non-linear isotonic (monotonic increasing) function of p_{it} and transforms it into a linear additive function. φ could typically be a log or logit function.

Although φ preserves the relative ordering of the probabilities, *i.e.*,

$$p_i > p_j \quad \Leftrightarrow \quad \varphi(p_i) > \varphi(p_j)$$

generally, it does not preserve the relative scale between probabilities, *i.e.*,

$$\frac{\varphi(p_i)}{\varphi(p_j)} \neq \frac{p_i}{p_j}$$

We can mitigate this issue by scaling to preserve the scale relationships between iterations,

$$\varphi(p_{ic}) \rightarrow \frac{\text{value}(p_{ic})}{\text{value}\big(\varphi(p_{ic})\big)} \varphi(p_{ic}) \tag{B.127}$$

where $\text{value}(\cdot) = \text{sgn}(\cdot)|\cdot|$, $\varphi(p_{ic}) = g_{ic}$ is will be expanded to fully express the weights of evidence for the one-*vs.*-none comparison, and $\text{value}\big(\varphi(p_{ic})\big)$ is its value sum, the total weight of evidence. The explicit value $\text{value}(p_{ic})$ is obtained from the probability distribution at inference, so it is also known. Note that even though the full dimensional weights of evidence is linearly additive, the weights for the base classifier dimensions (*e.g.*, input features) are actually in the transformed space, unlike the weights for the iteration dimension which are in the untransformed space. Consequently, the weights across dimensions are not strictly comparable, but this discrepancy may be unimportant to lay end-user, and this scaling may provide a sufficiently reasonable explanation.

Notice that even if φ is a linear transformation, the factor $\frac{\text{value}(p_{ic})}{\text{value}(\varphi(p_{ic}))}$ will not affect the relative scale of probabilities between iterations, and will even normalize each g_{ic} to probabilities. So, we can equally apply this scaling to all forms of g_{ic}.

B.20.3 BASE EXPLANATIONS BEFORE ENSEMBLE

Recall that some explainers do not provide a weights of evidence explanation g_i, such that $g_i = \{\emptyset\}$ (*e.g.*, explanations for SVMs). In such cases, we will not be able to obtain an ensemble explanation using Equation (B.126). First, we simplify Equation (B.126) to

$$g_i = \sum\nolimits_{c=1}^{C} \lambda_{ic} \frac{g_{ic}}{\text{value}(g_{ic})} \equiv \sum\nolimits_{c=1}^{C} \lambda_{ic} \tag{B.128}$$

where we neglect weights of evidence of the lower-level dimensions in g_{ic}. Next, we consider the weights of evidence explanation for a pairwise comparison, Equation (B.37):

$$\Delta g_{ij} = g_i - g_j \equiv \sum\nolimits_{c=1}^{C} \left(\lambda_{ic} - \lambda_{jc}\right) \tag{B.129}$$

Similarly to Equation (B.125), we consider normalizing the weights of evidence for a pairwise comparison:

$$\frac{\Delta g_{ic}}{\text{value}(\Delta g_{ic})} = 1 \tag{B.130}$$

and substitute Equation (B.130) into Equation (B.129) to get a weights of evidence explanation for pairwise comparison:

$$\begin{aligned} \Delta g_{ij} &\equiv \sum\nolimits_{c=1}^{C} \left(\lambda_{ic} - \lambda_{jc}\right) \frac{\Delta g_{ic}}{\text{value}(\Delta g_{ic})} \\ &= \sum\nolimits_{c=1}^{C} \sum\nolimits_{r=0}^{n} \Delta \bar{\bar{f}}_{ijcr} \end{aligned} \tag{B.131}$$

where

$$\Delta \bar{\bar{f}}_{ijcr} = \frac{\left(\lambda_{ic} - \lambda_{jc}\right)}{\text{value}(\Delta g_{ic})} \Delta f_{ijcr}$$

With Equation (B.129) we will still be able to generate Why and Why Not explanations. This produces the explanation of the pairwise comparison of each base classifier, Δf_{ijcr}, and collates them to produce the ensemble explanation. Since Equation (B.129) can be used more generally than Equation (B.126), we shall use it to derive subsequent explanations for ensemble classifiers.

B.20.4 SINGLE TYPE OF BASE CLASSIFIERS

Single level why:

$$\Delta g_{i\forall} = \frac{1}{m}\sum_{j=1}^{m} \Delta g_{ij} = \frac{1}{m}\sum_{j=1}^{m}\sum_{r=0}^{n} \Delta f_{ijr} \geq 0$$

In Sections B.21 and B.22, we describe explainers for two popular ensemble classifiers, Bootstrap Aggregation (Bagging) and AdaBoost.M1, respectively.

B.20.5 DIFFERENT TYPE OF BASE CLASSIFIERS

Some meta-classifiers (*e.g.*, Democratic Co-Learning [Zhou and Goldman, 2004]) use base classifiers of different types (*e.g.*, decision tree, naïve Bayes, kNN). These can lead to some inconsistency issues when generating a coherent weights of evidence explanation:

Number of dimensions of weights of evidence. Explainers may provide explanations in a different number of dimensions. For example, the naïve Bayes explainer (Section B.15) provides explanations only in terms of input features, x_r, the kNN explainer (Section B.19) provides explanations in terms of nearest neighbor and input features, x_{kr}, and the HMMs explainer (Section B.16) provides explanations in terms of time step and input features, x_{tr}. One simple way to standardize the number of dimensions is to just *reduce* the explanations to the dimension of input features. This is done using the `DimensionReducer` reducer component (see Section 6.11.2) that can collate weights of evidence to just the input feature dimension.

Dimension number space. Another issue is whether a non-linear transformation was used to derive the weight of evidence explanation in a base explainer. For example, log transforms were required for the naïve Bayes and decision tree explainers, and logit transforms were required for the logistic regression and multiclass linear SVM explainers, but only a linear transform was required for the kNN explainer. A consequence is the relative difference in magnitude of each weight of evidence.

Need to make each base classifier consistent in:

- Dimensions of weights of evidence
- Dimension space (*e.g.*, whether log-transformed)

B.20.5.1 STANDARDIZATION OF BASE WEIGHTS OF EVIDENCE

Need to be more stringent than Equation (B.126). Other than just normalize the total weights of evidence, we would like to normalize the *spread* of each weight of evidence. Equation (B.126) makes the mean (average) of atomic weights of evidence equal to $\frac{1}{n}$. Furthermore, we assume that each atomic weight of evidence will be of comparable magnitudes across classifiers. As such, the mean and standard deviation of the weights of evidence follow the distribution $\sim N\left(\frac{1}{n}, 1\right)$. Consider the rth atomic weight of evidence, f_r. The transformation for normalizing a distribution, $\sim N(0,1)$, is

$$N(0,1): \quad f_r \leftarrow \frac{f_r - \mu}{\sigma}$$

where μ is the average of the weights of evidence and σ is its standard deviation. Since we want to normalize and then make the mean $\frac{1}{n}$, we use the transformation

$$\text{standardize}(g, f_r) = N\left(\frac{1}{n}, 1\right): \quad f_r \leftarrow \frac{f_r - \mu}{\sigma} + \frac{1}{n}$$

Note that atomic weights that were originally zero will be biased toward a positive value.

We present the weights of evidence explanation as:

$$\Delta g_j = \sum\nolimits_{c=1}^{C} \sum\nolimits_{r=0}^{n} \Delta \bar{\bar{\bar{f}}}_{ijcr} \tag{B.132}$$

where

$$\Delta \bar{\bar{\bar{f}}}_{ijcr} = \left(\lambda_{ic} - \lambda_{jc}\right) \text{standardize}\left(\Delta g_{ijc}, \Delta f_{ijcr}\right)$$

$$\text{standardize}\left(\Delta g_{ijc}, \Delta f_{ijcr}\right) = \frac{\Delta f_{ijcr} - \mu_{ijc}}{\sigma_{ijc}} + \frac{1}{n}$$

$$\mu_{ijc} = \frac{1}{n}\sum\nolimits_{r=0}^{n} \Delta f_{ijcr}\,, \quad \sigma_{ijc} = \sqrt{\frac{1}{n}\sum\nolimits_{r=0}^{n} \left(\Delta f_{ijcr} - \mu\right)^2}$$

Due to the additional assumption of equal variance for atomic weights of evidence across base explanations, we do not use this normalization technique for ensemble classifiers that use a single type of base classifier.

B.21 BOOSTRAP AGGREGATION (BAGGING) EXPLAINER

Bootstrap aggregation (bagging) [Breiman, 1996] is an ensemble classification algorithm that can improve the classification accuracy of base classifiers. It takes a training dataset, D, and creates C new training sets by sampling examples from D with replacement. This method is called boostrap sampling. It then trains C versions of the base classifier, one classifier for each new training set. Overall inference is done by averaging the inference of each of the base classifiers:

$$p_i = \frac{1}{Z}\sum\nolimits_{c=1}^{C} p_{ic} \tag{B.133}$$

where p_{ic} is the probability of the cth classifier inferring the ith class, and Z is a normalization constant.

Using Equation (B.126), we can express the weights of evidence for inferring the ith class in terms of classifiers:

$$g_i = \sum\nolimits_{c=1}^{C} p_{ic} \frac{g_{ic}}{\text{value}(g_{ic})} \tag{B.134}$$

where g_{ic} is the linearly separable weights of evidence expression for the cth base classifier and $\text{value}(g_{ic}) = \text{sgn}(g_{ic})|g_{ic}|$ is the value of the total weights of evidence.

In terms of a pairwise classification, the weights of evidence explanation is:

$$\Delta g_{ij} = \sum\nolimits_{c=1}^{C} \left(p_{ic} - p_{jc}\right) \frac{\Delta g_{ijc}}{\text{value}\left(\Delta g_{ijc}\right)} \tag{B.135}$$

To illustrate the atomic weights of evidence for several bagged classifiers, we work out their expressions in the next sections, though they are automatically generated using Equation (B.134).

B.21.1 RANDOM FOREST (BAGGED RANDOM TREES)

Formatting the weights of evidence explanation for the Random Tree classifier within an ensemble classification, Equation (B.109) becomes:

$$g_{ic} = \log(p_{ic}) = \sum\nolimits_{r=0}^{n} \sum\nolimits_{\rho=0}^{\eta_c} \log(p_{ic\rho}) [\![x_r \in X_{c\rho}]\!] \tag{B.136}$$

where η_c is the length of the inferred trace of the cth classifier.

Substituting Equation (B.136) into Equation (B.134), we get

$$g_i = \sum\nolimits_{c=1}^{C} p_{ic} \frac{g_{ic}}{\text{value}(g_{ic})} = \sum\nolimits_{c=1}^{C} \sum\nolimits_{r=0}^{n} f_{icr} \tag{B.137}$$

where

$$f_{icr} = \frac{p_{ic}}{\log(p_{ic})} \sum\nolimits_{\rho=0}^{\eta_c} \log(p_{ic\rho}) [\![x_r \in X_{c\rho}]\!]$$

B.21.2 RANDOM (BAGGED) NAÏVE BAYES

Formatting the weights of evidence explanation for naïve Bayes within an ensemble classification, Equation (B.93) becomes:

$$g_{ic} = \log(p_{ic}) = \sum\nolimits_{r=0}^{n} \log(p_{icr}) \tag{B.138}$$

Substituting Equation (B.138) into Equation (B.134), we get

$$g_i = \sum\nolimits_{c=1}^{C} p_{ic} \frac{g_{ic}}{\text{value}(g_{ic})} = \sum\nolimits_{c=1}^{C} \sum\nolimits_{r=0}^{n} f_{icr} \tag{B.139}$$

where

$$f_{icr} = \frac{p_{ic}}{\text{value}\big(\log(p_{ic})\big)} \log(p_{icr})$$

B.21.2.1 BAGGED LINEAR SVM

Formatting the weights of evidence explanation for multiclass linear SVM within an ensemble classification, Equation (B.90) becomes:

$$\Delta g_{ijc} = \text{logit}(\mu_{ijc}) = \sum_{r=0}^{n} \Delta f_{ijcr} \tag{B.140}$$

where

$$\Delta f_{ijcr} = \omega_{ijcr} x_r$$

$$\omega_{ijcr} = \begin{cases} \kappa_{ijc}\beta_{ijc}w_{ijcr} & \text{, if } r > 0 \\ \kappa_{ijc}\beta_{ijc}w_{ijc0} + \kappa_{\text{ijc}}\gamma_{ijc} & \text{, if } r = 0 \end{cases} \qquad \kappa_{ijc} = \frac{\text{logit}(\mu_{ijc})}{\text{logit}(r_{ijc})}$$

Substituting Equation (B.140) into Equation (B.135), we get

$$\Delta g_{ij} = \sum_{c=1}^{C} (p_{ic} - p_{jc}) \frac{\Delta g_{ijc}}{\text{value}(\Delta g_{ijc})} = \sum_{c=1}^{C} \sum_{r=0}^{n} \Delta f_{ijcr} \tag{B.141}$$

where

$$\Delta f_{ijcr} = \frac{p_{ic} - p_{jc}}{\text{value}(\Delta g_{ijc})} \omega_{ijcr} x_r$$

B.22 ADABOOST.M1 EXPLAINER

AdaBoost.M1 is the multiclass extension of Discrete AdaBoost [Freund and Shapire, 1996], and this was extended for confidence-rated predictions in Real AdaBoost [Shapire and Singer, 1999]. We will not cover in detail the algorithm, and defer the interested reader to the source papers. AdaBoost.M1 defines the output inference of the meta-classification in terms of a weighted sum of the output inferences of $c$iterations of base classifier, *i.e.*,

$$h_i = \sum_{c=1}^{C} \alpha_c h_{ic} \tag{B.142}$$

where h_{ic} represents the learned function of whether the cth classifier iteration infers the ith class value, and α_c is the learned weight for the cth iteration.

To obtain the probability distribution over class values, we will need to calibrate the output of the boosted model (Equation (B.142)). Niculescu-Mizil and Caruana [2005] compared multiple calibration methods, and found that a Logistic correction was more accurate than using the raw weights, α_t, because boosting can be viewed as an additive logistic regression model [Friedman, Hastie, and Tibshirani, 2000]. This transformation applies an inverse logit transform (or expit or sigmoid) and takes the form:

$$p_i \propto \text{logit}^{-1}(2h_i) = \frac{1}{1 + \exp(-2h_i)}$$

$$\text{logit}(p_i) \propto h_i = \sum_{c=1}^{C} \alpha_c h_{ic}$$

The Weka toolkit performs a simpler calibration by applying an exponential transform instead:

$$p_i \propto \exp(h_i)$$

$$\log(p_i) \propto h_i = \sum_{c=1}^{C} \alpha_c h_{ic}$$

Similar to our previous treatments, our explanation will actually be about a monotonic transform on the probability of inference, rather than the actual probability distribution.

The Weka implementation of AdaBoost.M1 takes another simplification by defining

$$h_{ic} = [\![y_c = i]\!] \tag{B.143}$$

where we use the Kronecker delta notation where

$$[\![y_c = i]\!] = \begin{cases} 1 & \text{, if } c\text{th classifier infers } i\text{th class} \\ 0 & \text{, otherwise} \end{cases}$$

Substituting Equation (B.143) into Equation (B.142) gives

$$h_i = \sum_{c=1}^{C} \alpha_c [\![y_c = i]\!] = \sum_{c \in C_i} \alpha_c \tag{B.144}$$

where C_i is the set of iterations where the classifier inferred the ith class value. This discards any information we have regarding how underlying factors (*e.g.*, input feature values, class values) that influenced the base classifications. Nevertheless, this can help to simplify the explanations.

Using Equation (B.126), we can express the weights of evidence for inferring the ith class in terms of iterations:

$$g_i = \sum_{c=1}^{C} \alpha_c [\![y_c = i]\!] \frac{g_{ic}}{\text{value}(g_{ic})} \tag{B.145}$$

where g_{ic} is the Weights of Evidence for a one-*vs.*-none comparison explanation of the base classifier on the cth iteration, and $\text{value}(g_{ic}) = \text{sgn}(g_{ic})|g_{ic}|$ is the value of the total weights of evidence. Notice that even if we had to transform p_{ic} to derive g_{ic}, this does not influence the resultant explanation at this level, since the relative ranking of inferring each class is preserved. Note that the iteration dimension c may not be particularly informative, so this should be aggregated away before presenting to the end-user.

In terms of a pairwise classification, the weights of evidence explanation is:

$$\Delta g_{ij} = \sum_{c=1}^{C} \lambda_c \frac{\Delta g_{ijc}}{\text{value}(\Delta g_{ijc})} \tag{B.146}$$

where

$$\lambda_c = \alpha_c([\![y_c = i]\!] - [\![y_c = j]\!]) = \begin{cases} \alpha_c & \text{, if } c\text{th classifier infers } i\text{th class} \\ -\alpha_c & \text{, if } c\text{th classifier infers } j\text{th class} \\ 0 & \text{, otherwise} \end{cases}$$

To give an idea of what the atomic weights of evidence represent for several boosted classifiers, we work out their expressions in the next sections, though they are automatically generated using Equation (B.145).

B.22.1 BOOSTED DECISION STUMPS

Formatting the weights of evidence explanation for decision stumps within an ensemble classification, Equation (B.102) becomes:

$$g_{ic} = \sum_{r=0}^{n} p_{ic} [\![r_c = \tilde{r}_c]\!] \tag{B.147}$$

where r_c is the rth input feature of the th classifier, $\tilde{r}_c$ is the input feature selected for inference, and $[\![r_c = \tilde{r}_c]\!]$ indicates that r is selected input feature for cth classifier.

Substituting Equation (B.147) into Equation (B.145), we get

$$g_i = \sum\nolimits_{c=1}^{C} \alpha_c \left\llbracket i = \max_i(p_{ic}) \right\rrbracket \frac{g_{ic}}{\text{value}(g_{ic})} = \sum\nolimits_{c=1}^{C} \sum\nolimits_{r=0}^{n} f_{icr} \tag{B.148}$$

where

$$\begin{aligned} f_{icr} &= \alpha_c \left\llbracket i = \max_i(p_{ic}) \right\rrbracket \frac{p_{ic}}{\text{value}(p_{ic})} \llbracket r_c = \tilde{r}_c \rrbracket \\ &= \begin{cases} \alpha_c & \text{, if } \left(i = \max_i(p_{ic})\right) \cap (r_c = \tilde{r}_c) \\ 0 & \text{, otherwise} \end{cases} \end{aligned}$$

B.22.1.1 BOOSTED DECISION TREE

Formatting the weights of evidence explanation for the Decision Tree classifier within an ensemble classification, Equation (B.109) becomes:

$$g_{ic} = \log(p_{ic}) = \sum\nolimits_{r=0}^{n} \sum\nolimits_{\rho=0}^{\eta_c} \log\left(p_{ic\rho}\right) \llbracket x_r \in X_\rho \rrbracket \tag{B.149}$$

where η_c is the length of the inferred trace of the cth classifier.

Substituting Equation (B.149) into Equation (B.145), we get

$$g_i = \sum\nolimits_{c=1}^{C} \alpha_c \left\llbracket i = \max_i(p_{ic}) \right\rrbracket \frac{g_{ic}}{\text{value}(g_{ic})} = \sum\nolimits_{c=1}^{C} \sum\nolimits_{r=0}^{n} f_{icr} \tag{B.150}$$

where

$$\begin{aligned} f_{icr} &= \frac{\alpha_c \left\llbracket i = \max_i(p_{ic}) \right\rrbracket}{\text{value}\left(\log(p_{ic})\right)} \sum\nolimits_{\rho=0}^{\eta_c} \log\left(p_{ic\rho}\right) \llbracket x_r \in X_{c\rho} \rrbracket \\ &= \begin{cases} \alpha_c \sum\nolimits_{\rho=0}^{\eta_c} \dfrac{\log\left(p_{ic\rho}\right)}{\log(p_{ic})} \llbracket x_r \in X_{c\rho} \rrbracket & \text{, if } i = \max_i(p_{ic}) \\ 0 & \text{, otherwise} \end{cases} \end{aligned}$$

B.22.1.2 BOOSTED KNN

Formatting the weights of evidence explanation for the kNN classifier within an ensemble classification, Equation (B.122) becomes:

$$g_{ic} = \sum_{k=1}^{J} \sum_{r=1}^{n} f_{ickr} \tag{B.151}$$

where

$$f_{ickr} = \omega_{ickr} [\![y_{ck} = i]\!]$$

$$\omega_{ickr} = \begin{cases} \dfrac{1}{n} & \text{No Weighting} \\ \dfrac{1}{n} - \delta_{ckr} & \text{Similarity Weighting, or} \\ & \text{Inverse Weighting}^{\dagger} \end{cases} \qquad \delta_{ckr} = \begin{cases} \Delta x_{ckr} & \text{Manhattan} \\ \hat{\gamma}\hat{\alpha}_r \Delta x_{ck,(r)} & \text{Euclidean}^{\dagger} \end{cases}$$

$$\Delta x_{ckr} = |x_{ir} - x_{ckr}|$$

Substituting Equation (B.151) into Equation (B.145), we get

$$g_i = \sum_{c=1}^{C} \alpha_c \left[\!\!\left[i = \max_i(p_{ic}) \right]\!\!\right] \frac{g_{ic}}{\text{value}(g_{ic})} = \sum_{c=1}^{C} \sum_{k=1}^{J} \sum_{r=1}^{n} f_{ickr} \tag{B.152}$$

where

$$f_{ickr} = \frac{\alpha_c \left[\!\!\left[i = \max_i(p_{ic}) \right]\!\!\right]}{\text{value}(g_{ic})} \omega_{ickr} [\![y_{ck} = i]\!]$$

B.23 ENSEMBLE DECISION TREES RULE TRACES EXPLAINER

Given the popularity of using decision trees in ensemble classifiers [?], we can present richer explanations of an ensemble of decision trees, where for each inference, there is a trace for each decision tree.

Extending Equation (B.11)

$$\begin{gathered}
\Gamma_i = \bigcap_{c=1}^{C} \bigcap_{\rho=0}^{\eta_c} X_{c\rho} \\
g_i = \sum_{c=1}^{C} \lambda_{ic} \frac{g_{ic}}{\text{value}(g_{ic})} = \sum_{c=1}^{C} \sum_{\rho=0}^{\eta_c} f_{icr}
\end{gathered} \tag{B.153}$$

where

$$f_{icr} = \frac{\lambda_{ic}}{\log(p_{ic})} \log(p_{ic\rho})$$

Note that each trace, η_c, may be of different length.

B.24 SUMMARY OF EXPLAINERS FOR INFERENCE MODELS

We summarize the two main types of explainers: Rule Trace and Weights of Evidence.

B.24.1 DNF RULE TRACE EXPLAINER

These explainer methods apply to Rules and Decision Trees after conversion to disjunctive normal form (DNF). We store rules as DNF trees, and separate based on which class value they infer. These DNF trees are stored in a map of DNF trees:

$$\Phi = \bigcup_{j=1}^{m} \psi_j$$

where

$$\psi_j = \bigcup_{v} \mathrm{\xi}_{\mathrm{jv}} \text{ is the DNF tree of traces that infer the } j\text{th class}$$

$$\mathrm{\xi}_{\mathrm{jv}} = \bigcap_{\mathrm{\rho}_{\mathrm{jv}}} \mathrm{X}_{\mathrm{jv\rho}} \text{ is a trace consisting of conditions } \mathrm{X}_{\mathrm{iv\rho}}$$

Explanation Types	**Rule Trace Explanations**
Why	$\psi_i \vert x = \bigcup_{v_i} \xi_{iv} [\![\xi_{iv} \vert x]\!] = \bigcup_{v_i} \bigcap_{\rho_{vk}} X_{jv\rho} [\![X_{iv\rho} \vert x]\!]$ where $[\![\xi_{iv} \vert x]\!] = \begin{cases} 1 & \text{, if } \xi_{iv} \text{ is true given inputs } x \\ 0 & \text{, otherwise} \end{cases}$ $\xi_{iv} \vert x = \bigcap_{\rho_{iv}} X_{iv\rho} \vert x = \bigcap_{\rho_{iv}} \bigcap_{r=1}^{n} X_{iv\rho} \vert x_r$ is whether trace ξ_{iv} is satisfied by inputs x $X_{iv\rho} \vert x_r = \begin{cases} \text{TRUE} & \text{, if } X_{iv\rho} \text{ is true given } x_r \\ \text{TRUE} & \text{, if } x_r \notin X_{iv\rho} \\ \text{FALSE} & \text{, otherwise} \end{cases}$
Why Not	$\overline{\psi_\jmath} \vert x = \bigcup_{v_j} \overline{\xi_{\jmath v}} \vert x = \bigcup_{v_j} \bigcup_{\rho_{jv}} \overline{X_{\jmath v\rho}} [\![\overline{X_{\jmath v\rho}} \vert x]\!]$ where $\overline{X_{\jmath v\rho}} \vert x = \begin{cases} \text{FALSE} & \text{, if } X_{iv\rho} \text{ is true given } x \\ \text{TRUE} & \text{, if } x_r \notin X_{iv\rho} \\ \text{TRUE} & \text{, otherwise} \end{cases}$
How To	$\psi_i = \Phi \cdot [\![i = j]\!] = \bigcup_{j=1}^{m} \psi_j [\![i = j]\!]$ $= \bigcup_{v_i} \xi_{iv} = \bigcup_{v_i} \bigcap_{\rho_{vk}} X_{jv\rho}$

B.24.2 Weights of Evidence Explainers

Explanation Types	**Rule Trace Explanations**
Why So (Absolute Why)	$g_i = \sum_{r \in R} f_{ir}$ where f_{ir} is the rth atomic weight of evidence that votes for the ith class of the set of all atomic factors, R.
Why Not (Pairwise Comparison)	$\Delta g_{ij} = \sum_{r \in R} \Delta f_{ijr}$ where Δf_{ijr} is the pairwise weight of evidence for inferring the ith instead of the jth class. For explainers where f_{ir} is defined, $\Delta g_{ij} \equiv g_i - g_j = \sum_{r \in R} (f_{ir} - f_{jr})$ $\Delta f_{ijr} \equiv f_{ir} - f_{jr}$
Why Best (Relative Why)	$\Delta g_{i\forall} = \sum_{j=1}^{m} \Delta g_{ij} = \sum_{j=1}^{m} \sum_{r \in R} \Delta f_{ijr}$ where weights of evidence consists of an extra dimension due to class value $j = [1, m]$.
How To	*Depends on inference model*

The following table describes how to generate the weights of evidence expression for specific inference models.

Inference Model		**Weights of Evidence Explanations**
Functional Regression	Linear Regression	$g = y = \sum_{r=0}^{n} f_r$ where $f_r = a_r x_r$
Function Classification	Multinomial Logistic Regression	$g_i = \mathrm{logit}(p_i) = \sum_{r=0}^{n} f_{ir}$ where $f_{ir} = \beta_{ir} x_r$ $\beta_{ir} = \begin{cases} \beta_{ir} & i < m, r > 0 \\ \beta_{ir} + \beta_{m0} & i < m, r = 0 \\ 0 & i = m, r > 0 \\ \beta_{m0} & i = m, r = 0 \end{cases}, \beta_{m0} \neq 0$ $x_r = \begin{cases} x_r & r > 0 \\ 1 & r = 0 \end{cases}$

Inference Model		**Weights of Evidence Explanations**
	Uncalibrated Multiclass Linear SVM	$\Delta g_{ij} = \sum_{r=0}^{n} \Delta f_{ir}$ where $\Delta f_{ijr} = \frac{\omega_{ijr}}{\text{value}(y_{ij})} x_r [\![y_{ij} > 0]\!]$ $\omega_{ijr} = \begin{cases} w_{ijr} & \text{, if } r > 0 \\ -b & \text{, if } r = 0 \end{cases}$
	Pairwise-coupled, Platt-calibrated Multiclass Linear SVM	$\Delta g_{ij} = \text{logit}(\mu_{ij}) = \sum_{r=0}^{n} \Delta f_{ir}$ where $\Delta f_{ijr} = \omega_{ijr} x_r$ $x_0 = 1$ **By Difference** $\omega_{ijr} = \begin{cases} \beta_{ij} w_{ijr} & \text{, if } r > 0 \\ \beta_{ij} w_{ij0} + \gamma_{ij} + \delta_{ij}, & \text{if } r = 0 \end{cases}$ $\delta_{ij} = \text{logit}(\mu_{ij}) - \text{logit}(r_{ij})$ **By Scale** $\omega_{ijr} = \begin{cases} \kappa_{ij} \beta_{ij} w_{ijr} & \text{, if } r > 0 \\ \kappa_{ij}(\beta_{ij} w_{ij0} + \gamma_{ij}), & \text{if } r = 0 \end{cases}$ $\kappa_{ij} = \frac{\text{logit}(\mu_{ij})}{\text{logit}(r_{ij})}$
	Multiclass Quadratic SVM	$\Delta g_{ij} = \sum_{q=0}^{n} \sum_{r=0}^{n} \Delta f_{ijqr}$ where $\Delta f_{ijqr} = \begin{cases} c^2 + \gamma_{ij} + \delta_{ij} & \text{, if } q = 0, r = 0 & \text{constant term} \\ 2c \sum_{s=1}^{v} a_{ijsr} x_{ijsr} & \text{, if } q = 0, r > 0 & \text{linear terms} \\ 0 & \text{, if } q > 0, r = 0 & \\ \sum_{s=1}^{v} (a_{ijsr} a_{ijsq}) x_{ijsr} x_{ijsq} & \text{, if } q > 0, r > 0 & \text{cross terms} \end{cases}$
Bayesian	Naïve Bayes	$g_i = \log(p_i) = \sum_{r=0}^{n} f_{ir}$ where $f_{ir} = \log(p_{ir})$ $p_{ir} = \begin{cases} P(y_i) & \text{, if } r = 0 \\ P(x_r \mid y_i) & \text{, if } r > 0 \end{cases}$ $p_i = P(y_i \mid \vec{x})$

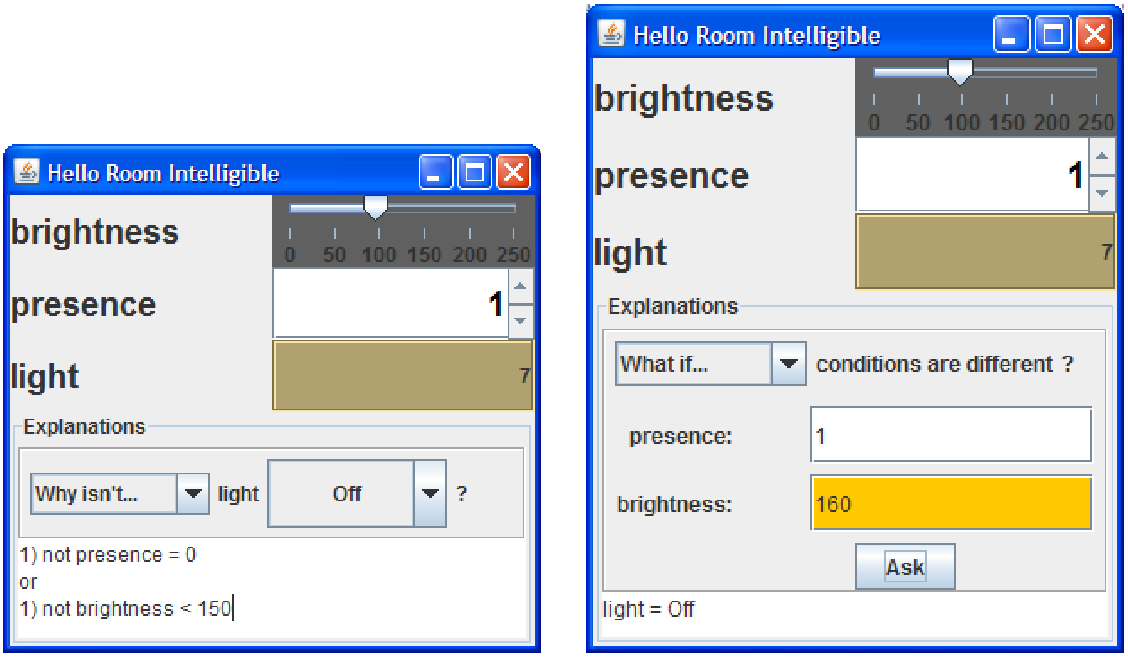
Inference Model		Weights of Evidence Explanations
	Hidden Markov Models (HMM)	$g_s \cong \sum_{t=1}^{T} \sum_{r=0}^{n} f_{str}$ where $f_{str} = \log(p_{str})$ $p_{str} = \begin{cases} \pi(s_1) & r = 0, t = 1 \\ A(s_t \mid s_{t-1}) & r = 0, t > 1 \\ \tilde{B}(x_{tr} \mid s_t) & r > 0, t > 1 \end{cases}$ Naïve HMM model, $\lambda = (\pi, A, \tilde{B})$
Decision Tree	Probabilistic Decision Tree	$g_i = \log(p_i) = \sum_{\rho=0}^{\eta} f_{i\rho}$ where ρ indexes the trace node, rather than input feature, $f_{i\rho} \cong \log(p_{i\rho})$
		$g_i = \sum_{r=0}^{n} f_{ir}$ where $f_{ir} = \sum_{\rho=1}^{\eta} \log(p_{ir}) \llbracket x_r \in X_\rho \rrbracket$
Similarity	k-Nearest Neighbors (kNN)	$g_i \cong \sum_{k=1}^{J} \sum_{r=0}^{n} f_{ikr}$ where $f_{ikjr} = \omega_{ikr} \llbracket y_i = y_k \rrbracket$ $\omega_{ikr} = \begin{cases} 1 \\ 1 - \delta_{ikr} \end{cases}$ No Weighting Similarity, or Inverse $\delta_{ikr} = \begin{cases} \Delta x_{ikr} & \text{Manhattan} \\ \gamma \alpha_r \Delta x_{ik,(r)} & \text{Euclidean} \end{cases}$ $\alpha_r = \sqrt{r} - \sqrt{r-1}$ $\gamma = \frac{2}{1 + \sqrt{\sum_{r=1}^{n} \alpha_r^2}}$ $\Delta x_{ikr} = x_{ir} - x_{kr}$ x_{ir} normalized For Euclidean distance, input feature ordered where $x_{(r)}$ is a permutation of $\{\lvert x_r \rvert\} = \langle \lvert x_1 \rvert, \dots, \lvert x_n \rvert \rangle$ such that $x_{(1)} \geq \cdots \geq x_{(n)}$ $\Delta x_{ik0} = 0$

Inference Model		**Weights of Evidence Explanations**
Ensemble	Bootstrap Aggregation (Bagging)	$g_i = \sum_{c=1}^{C} \frac{p_{ic}}{\text{value}(g_{ic})} g_{ic}$ $\Delta g_{ij} = \sum_{c=1}^{C} \frac{p_{ic} - p_{jc}}{\text{value}\left(\Delta g_{ijc}\right)} \Delta g_{ijc}$ where g_{ic} defined by base classifier, value(g_{ic}) represents the value of the evidence (*i.e.*, total evidence).
	AdaBoost.M1	$g_i = \sum_{c=1}^{C} \frac{\alpha_c \llbracket y_c = i \rrbracket}{\text{value}(g_{ic})} g_{ic}$ $\Delta g_{ij} = \sum_{c=1}^{C} \frac{\alpha_c (\llbracket y_c = i \rrbracket - \llbracket y_c = j \rrbracket)}{\text{value}\left(\Delta g_{ijc}\right)} \Delta g_{ijc}$ where g_{ic} defined by base classifier.

C Hello World Tutorial for Intelligibility Toolkit

This tutorial is an adaptation of the tutorial at http://www.contexttoolkit.org/?p=197 about how to add intelligibility to an existing context-aware application, HelloRoom. See the tutorial at http://www.contexttoolkit.org/?p=50 to learn how to build that application with the Context Toolkit.

This tutorial describes how to make a context-aware application **intelligible**, such that it can explain, what it did, why, and how it works. We will be using the components of the Intelligibility Toolkit. We will extend the HelloRoom context-aware application to make it provide explanations.



presence: 1

1) not presence = 0
or
1) not brightness < 150

presence: 1

brightness: 160

light = Off

Why Not explanation **What If explanation**

Figure C.1. Screenshots of the Hello Room application showing some explanations.

We will be reusing the classes we had defined in the HelloRoom tutorial, and extending the main application class, HelloRoom, with HelloRoomIntelligible:

```
public class HelloRoomIntelligible extends HelloRoom {

/** Intelligibility UI */
protected JPanel iui;

   public HelloRoomIntelligible() {
      super();

      // set new UI component
      iui =  new IntelligibleUI(enactor);
      iui.setVisible(false);
   }

   @Override
   public void enactorsReady() {
      super.enactorsReady();
      iui.setVisible(true);
   }

   ...

}
```

We retain the original HelloRoomUI panel that contains the GUI elements to change brightness and presence, and see the corresponding change in light level. We add IntelligibleUI to provide a JPanel to show GUI elements for asking for and viewing explanations. We override enactorsReady() to also set this to be visible only when the enactors are ready. It is in the implementation of IntelligibleUI that we will employ the various components of the Intelligibility Toolkit.

```
public class IntelligibleUI extends JPanel {

   // ... instance fields

   public IntelligibleUI(final Enactor enactor) {
      super();
      setLayout(new BorderLayout());
      setBorder(BorderFactory.createTitledBorder("Explanations"));

      ...
   }

}
```

For the rest of the code in the constructor, we set up the components for the intelligibility features:

```
// UI for obtaining queries from the user
queryPanel = new QueryPanel(enactor, true);
add(queryPanel, BorderLayout.NORTH);

// reducer for showing only brightness and presence in explanations
creducer = new FilteredCReducer("brightness", "presence", "light");

// presenter for rendering explanations
presenter = new StringPresenter(enactor);

// UI for showing explanation
explanationArea = new JTextArea();
add(explanationArea, BorderLayout.CENTER);
```

We use the utility class `QueryPanel` provided in the toolkit to get a JPanel that provides a GUI for getting a `Query` from the user. We then create a conjunction reducer, `FilteredCReducer`, to simplify explanations that will be generated from the query. This particular reducer only provides explanations about the attributes with names from the array {"brightness", "presence", "light"}. We instantiate `StringPresenter`, a simple Presenter to render explanations into a String representation. Finally, we create a GUI text area to show the explanation in. This will be populated when an explanation is generated.

Next, we need to listen for when queries are created by the `QueryPanel`, and we do this by adding a `QueryListener` which implements `queryInvoked(Query)`:

```
queryPanel.addQueryListener(new QueryListener() {
   @Override
   public void queryInvoked(Query query) {
      // generate explanation
      Explanation explanation = enactor.getExplainer().getExplanation(query);
      System.out.println("explanation = " + explanation);

      // reduce
      explanation = creducer.apply(explanation);

      // render
      String explanationText = presenter.render(explanation);

      explanationArea.setText(explanationText);
   }
});
```

Here, we take the Query that is passed and put it into an `Explainer` to get and Explanation. All Enactors come with a default Explainer depending on their underlying model. The content of the explanation is an `Expression` in Disjunctive Normal Form (`DNF`), and this is often overly complicated for end-users to interpret. So we apply our previously instantiated reducer to reduce the explanation. With the final form of the explanation, we can render it to human-readable form with our presenter to get an explanation text, which we display in the text area.

So that is all that is needed to add intelligibility capabilities to the HelloRoom application. The final step is just to launch the application in a JFrame:

```java
public static void main(String[] args) {
   ContextModel.startDiscoverer();

   HelloRoomIntelligible app = new HelloRoomIntelligible();
   app.start();

   JFrame frame = new JFrame("Hello Room Intelligible");
   frame.add(app.ui, BorderLayout.NORTH);
   frame.add(app.iui, BorderLayout.CENTER);
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   frame.setSize(new Dimension(320, 360));
   frame.setLocationRelativeTo(null); // center of screen
   frame.setVisible(true);
}
```

Source code for this tutorial is provided with the Context Toolkit distribution with the source code for HelloRoom.

D LAKSA INFERENCE MODELS

This appendix describes the reasoning and inference models used in Laκsa (Chapter 7) and Laκsa2 (Chapter 9).

D.1 AVAILABILITY

Laκsa infers the user's availability by applying rules (*e.g.*, see Personal Availability Rules in Appendices E.1 and Rules in I.1) based on lower-level contexts. Multiple rules may be triggered simultaneously, prioritized by availability from Unavailable first and Available last.

D.2 PLACE

Laκsa infers whether the user is at a particular Place (denoted by semantic name, *e.g.*, Home, Office) by sensing the following attributes through:

- Measuring the user's location Coordinates (`Latitude, Longitude`) via GPS or Network on the Android phone,
- Estimating the `Accuracy` of the location fix as returned by `Location.getAccuracy()`,
- Retrieving `Places` that the user had saved and which are near to the coordinates; these are modeled as circular regions ("bubbles") with center position selected by the user (*e.g.*, via a map interface) and radius as defined by the user.

In Laκsa (version 1), the user's location is modeled as a circular region "bubble" with the Coordinates as its center and the Accuracy as its radius. We define this as the "user bubble" and the circular region for a Place as the "place bubble." The user is inferred at a Place if the user bubble overlaps with the place bubble (see Figure D.1).

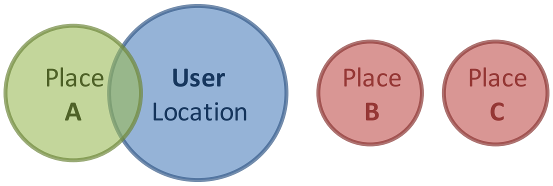


Figure D.1. Bubbles model used in Laκsa of the sensed user location and three semantic places A, B, and C. This indicates that the user is inferred at A because of the overlap.

In Laκsa2, the user bubble is defined as a Gaussian area, where we have interpreted the location Accuracy as the 50% circular error probability (CEP)[7,8] [CEP]. This indicates the circular region within which there is a 50% probability the user is located. By mapping the location accuracy to 50% CEP, we can model the Gaussian area and derive the accuracy bounds for other CEP values (see Figure D.2).

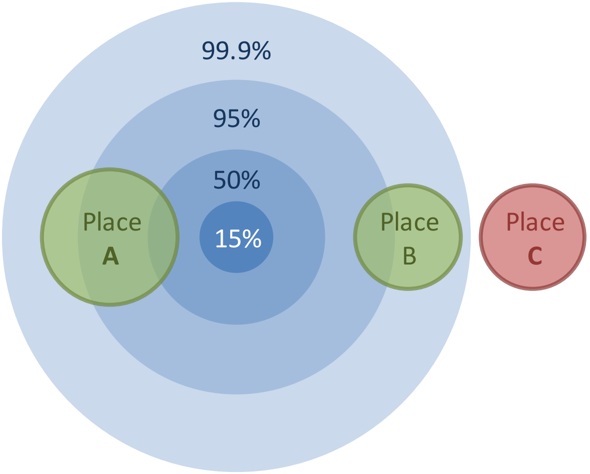


Figure D.2. Bubbles model used in Laκsa2 of the sensed user location and three semantic places A, B, and C. The user location is modeled as a Gaussian area from the location coordinates. Concentric circles indicate location accuracy bounds for: 15%, 50%, 95%, 99.9% CEP. This indicates that the user is inferred at A (most likely) followed by B, because of the relative amounts of overlapping areas.

We can also determine the inference certainty of the user being at various places by calculating the area overlap of the user Gaussian area with the place circle. For simpler computation on a mobile phone, Laκsa2 approximates the Gaussian area as four concentric rings, representing 15%, 50 – 15 = 35%, 95 – 50 = 45%, 99.9 – 95 ≈ 5% probability of the user being each ring, respectively.

[7] Circular Error Probable. http://en.wikipedia.org/wiki/Circular_error_probable. Retrieved 27 April 2012.

[8] GPS Accuracy. http://www.kowoma.de/en/gps/accuracy.htm. Retrieved 27 April 2012.

D.3 Sound Activity

Laκsa infers sound activity by

1. Sampling from the microphone signal,
2. Extracting features from the signal, and
3. Applying machine learning to distinguish between different types of sound

We perform sampling and feature extraction using the methods similar to similar to SoundSense [Lu *et al*., 2009]. However, the inference uses the simpler naïve Bayes classifier which is explainable with the Intelligibility Toolkit.

D.3.1 Audio Sampling from Microphone

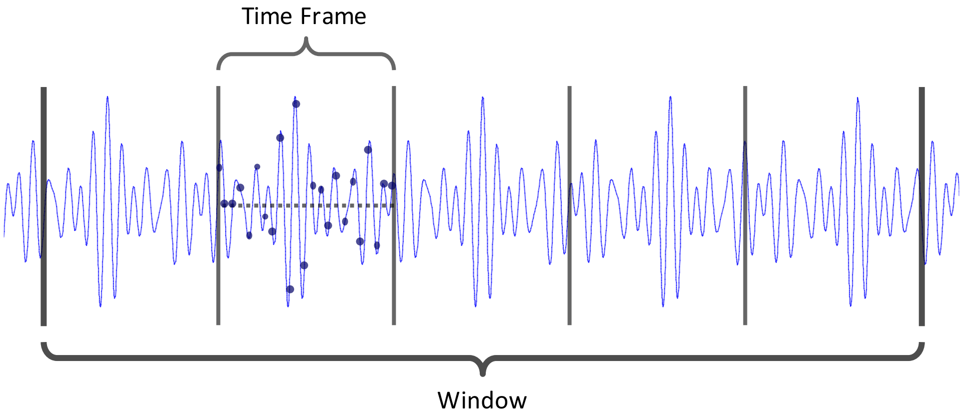


Figure D.3. Sampling of audio signal partitioned into Windows and Time Frames.

D.3.1.1 Time Series Sampling

Laκsa samples the audio signal at a rate of 8000 samples/sec in time windows of 1.6 seconds. After which, the sample is pre-processed to extract features and classification for sound inference is performed. Each window is divided into 25 time frames (see Figure D.3) each of duration 64 ms and containing 512 samples. For each time frame, we denote the tth sample as x_t. Independent of CPU time for feature extraction and classification, Laκsa requires at least 1.6 seconds to sample the audio. Table D.2 summarizes the sampling characteristics for audio.

D.3.1.2 FAST FOURIER TRANSFORM TO OBTAIN FREQUENCY SPECTRUM

We perform a Fast Fourier Transform (FFT) to obtain spectral information of the signal. Without going into details, the transformation produces:

$$\sum_{t=1}^{N} x_t \xrightarrow{\text{FFT}} \sum_{k=1}^{M} X_k \qquad \text{(D.1)}$$

where $X_k = a_k + ib_k$ is a complex number representing the kth bin in the frequency series, $\mathrm{Re}(X_k) = a_k$ is its real component and $\mathrm{Im}(X_k) = b_k$ is its imaginary component. Note that the frequency series is only half as long ($M = \mathrm{N}/2$) as the time series because the latter consists of only real numbers and the frequency series is consequently symmetric.

By considering the FFT output of complex numbers, we can calculate the amplitude and phase of each frequency bin:

$$\text{Frequency amplitude: } A_k = |X_k| = \sqrt{a_k^2 + b_k^2}$$

$$\text{Frequency phase: } \phi_k = \arg(X_k) = \arctan(b_k/a_k)$$

For spectral features, we normalize the amplitudes of each frequency bin: $p_k = A_k / \sum_{k=1}^{N/2} A_k$.

The characteristics[9] of the frequency spectrum is summarized in Table D.2.

Domain	Characteristic	Formula	Value
Time Series	Sampling rate	S	8000 samples/sec
	Sample duration (time frame length)	T	0.064 sec (64 ms)
	Number of samples	$N = S \times T$	512 samples
	Number of time frames / window	n	25 frames/window
	Window length	$n \times T$	1.6 sec
Frequency	Number of bins	$M = N/2$	256 bins
	Maximum resolvable frequency	$F_{max} \equiv F_{Nyquist} = S/2$	4000 Hz
	Frequency Resolution	$dF = S/N \equiv F_{max}/M$	15.625 Hz

Table D.1. Sampling characteristics from the audio signal.

[9] FFT Fundamentals. http://zone.ni.com/reference/en-XX/help/372416B-01/svtconcepts/fft_funda/. Retrieved 28 April 2012.

D.3.2 FEATURES EXTRACTED FROM AUDIO SIGNAL

Laĸsa extracts several features from the audio signal by counting and statistical measures from the sample measures of time frames and the window. Several features are based on time series data, while others are based on the FFT spectrum.

D.3.2.1 POWER

Power, P, is a time frame measure which indicates the energy per unit time in the audio signal. This can be represented as the root mean square (RMS), x_{rms}, of the signal in the time frame or alternatively represented in a decibel scale. We use the dBFS[10] scale calibrating $x_{rms} = x_{max} = 32767$ to $P_{max} = 0$ as the maximum value and mapping $x_{rms} = 1$ to $P_{min} \gtrsim -100$. Note that we cannot use $x_{rms} = 0$ to calibrate the dB scale, since the log is asymptotic. In Android, the audio signal is read as an array of `short` numbers, so the maximum signed value is $x_{max} = 32767$. The power of the audio signal for the time frame is:

$$P = \psi \log \frac{x_{rms}}{x_{max}} \tag{D.2}$$

where

$$x_{rms} = \frac{1}{N} \sqrt{\sum_{t=1}^{N} x_t^2}$$

We define $\psi = 20$, such that

$$P = \begin{cases} 0 & ,\text{when } x_{rms} = x_{max} = 32767 \\ -90.3 & ,\text{when } x_{rms} = 1 \end{cases}$$

Since Power is a time frame measure, for frame f, we denote its power as P_f.

D.3.2.2 LOW-ENERGY FRAME RATE

Low-Energy Frame Rate (LEFR) is a window measure which indicates the proportion of time frames with energy (power) *less than half* of the mean energy for all time frames in the window. A higher LEFR value indicates more frames that have low energy compared to the whole window. Speech typically has high LEFR values because of pauses between words. LEFR is calculated as:

[10] Decibels relative to Full Scale [Price, 2007]

$$LEFR = \frac{1}{n}\sum\nolimits_{f=1}^{n} \left[\!\left[P_f < \frac{\bar{P}}{2} \right]\!\right] \tag{D.3}$$

where

$\bar{P} = \frac{1}{n}\sum\nolimits_{f=1}^{n} P_f$ is the mean power in the window, and

$\left[\!\left[P_f < \frac{\bar{P}}{2} \right]\!\right] = \begin{cases} 1 & \text{, if } P_f < \bar{P}/2 \\ 0 & \text{, otherwise} \end{cases}$, where we use the Kronecker delta ⟦·⟧

D.3.2.3 ZERO CROSSING RATE

Zero Crossing Rate (ZCR) is a time frame measure which indicates the number of times the signal crosses the zero value. A crossing is measured when consecutive samples have different signs, *i.e.*, $x_t \cdot x_{t-1} < 0$. We calculate ZCR as:

$$ZCR = \frac{1}{2}\sqrt{\sum\nolimits_{t=1}^{N} [\![x_t \cdot x_{t-1} < 0]\!]} \tag{D.4}$$

D.3.2.4 SPECTRAL FLUX

Spectral flux (SF) is a spectral time frame measure indicating the variability of the frequency profile between consecutive frames. For the tth time frame in the window, its spectral flux is:

$$SF(t) = \frac{1}{M}\sqrt{\sum\nolimits_{k=1}^{M} \left(p_k(t) - p_k(t-1)\right)^2} \tag{D.5}$$

where $p_k(t)$ is the normalized amplitude of the kth frequency bin of the tth time frame.

D.3.2.5 SPECTRAL ROLLOFF

Spectral rolloff (SRF) is a spectral time frame measure indicating the frequency at which a high threshold of the spectrum energy is below.

$$SRF = f_\kappa = \underset{\kappa<M}{\operatorname{argmin}} \left[\left(\sum\nolimits_{k=1}^{\kappa} p_k \right) > \text{threshold} \right] \tag{D.6}$$

where f_κ is the frequency of the κth bin. For Laκsa, we chose as 93% as the threshold, similar to SoundSense [Lu *et al.*, 2009].

D.3.2.6 SPECTRAL CENTROID

Spectral Centroid (SC) is a spectral time frame measure indicating the "middle" of frequencies in the spectrum in the time frame:

$$SC = f_{\bar{k}} \quad \text{, where } \bar{k} = \frac{\sum_{k=1}^{M} k \cdot p_k^2}{\sum_{k=1}^{M} p_k^2} \tag{D.7}$$

where we also denote the centroid with $\bar{k}$.

D.3.2.7 BANDWIDTH

The Bandwidth, BW, is a spectral time frame measure which indicates the "spread" of salient frequencies in the time frame:

$$BW = dF \cdot k_{BW} \quad \text{, where } k_{BW} = \sqrt{\frac{\sum_{k=1}^{M} \left(k - \bar{k}\right)^2 \cdot p_k^2}{\sum_{k=1}^{M} p_k^2}} \tag{D.8}$$

where $\bar{k}$ is the spectral centroid and dF is the frequency resolution.

D.3.2.8 NORMALIZED WEIGHTED PHASE DETECTION

Normalized Weighted Phase Detection (NWPD) is a spectral time frame feature which considers the extent that all frequencies are in phase; if they are very out of phase, then they will destructively interfere with one another and reduce the energy of the signal. We calculate NWPD as:

$$NWPD = \sum_{k=1}^{M} p_k \cdot \ddot{\phi}_k \tag{D.9}$$

where $\ddot{\phi}_k = \frac{d^2\phi_k}{dt^2}$ is the second derivative of the phase of the kth frequency bin.

D.3.2.9 RELATIVE SPECTRAL ENTROPY

Relative Spectral Entropy (RSE) is a spectral time frame indicating how the 'shape' (entropy) of the spectrum changes over time; it helps to differentiate speech from other sounds. We calculate it as:

$$RSE(t) = -\sum_{k=1}^{M} p_k(t) \log \frac{p_k(t)}{m_k(t-1)} \tag{D.10}$$

where $m_k(t)$ is the mean amplitude across time frames of the kth frequency bin for the tth time frame, which we approximate only with the current and previous amplitudes:

$$m_k(t) = \begin{cases} 0.9m_k(t-1) + 0.1p_k(t) & \text{, if } t > 0 \\ p_k(0) & \text{, if } t = 0 \end{cases}$$

D.3.2.10 MEL-FREQUENCY CEPSTRAL COEFFICIENTS

Mel-Frequency Cepstral Coefficients (MFCCs) are features further extracted from the FFT spectrum which are popular for speech and speaker recognition. These coefficients can be derived from the following procedure (adapted from [Crittenden and Evans, 2008]):

1. Take the FFT of a time frame of the signal to get the FFT spectrum.
2. Define a set of C triangular overlapping filters in the mel frequency scale to quantify the power (energy) in various regions of the spectrum. This produces a set of mel 'bins'.
3. Take the logs of powers at each filtered region.
4. Take the Discrete Cosine Transform (DCT) of the log mel powers, as if the mel bins represent a signal. The result is a cepstrum.
5. The MFCCs are the amplitudes of the resulting cepstrum.

The mel frequency scale represents how people perceive sound frequency linearly with respect to mel frequency. Frequency can be transformed to the mel scale with:

$$m = 1127 \log_e \left(\frac{f}{700} + 1 \right) \tag{D.11}$$

Note that the maximum resolvable frequency is 4000 Hz which is Nyquist frequency given the sampling rate of 8000 samples/sec. The maximum mel frequency is thus $m_{max} = 2146$ mel.

Rather than transform the frequency spectrum into mel frequency, we can alternatively first define the triangular overlapping windows which we will use to filter the spectrum, inverse transform them into the frequency domain:

$$f = 700 \left(e^{\frac{m}{1127}} - 1 \right) \tag{D.12}$$

For Laκsa, we use $C = 12$ triangular filters to measure the power in various regions in the spectrum uniformly distributed along mel scale (see Figure D.4).

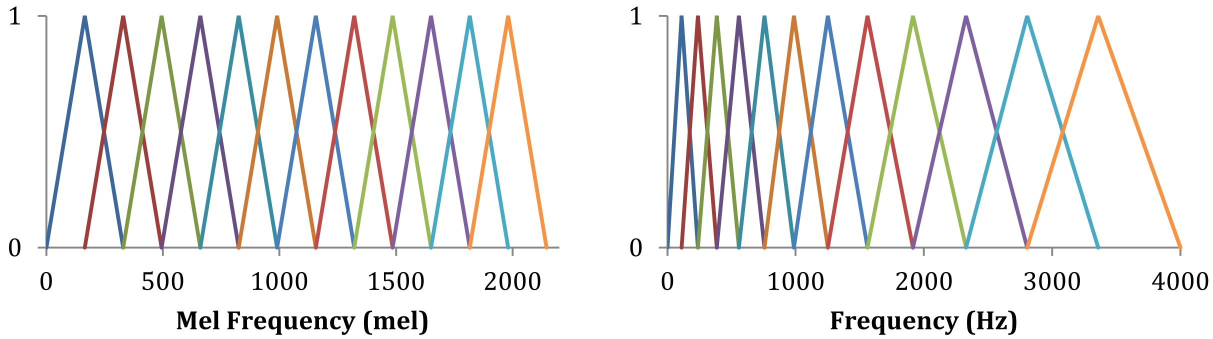


Figure D.4. Triangular overlapping filters applied to the frequency spectrum: symmetric in the Mel Frequency domain (Left) and distorted in the Frequency domain (Right).

In the mel frequency domain, the mid-point (apex) of each ith triangular filter is evenly spaced, *i.e.*,

$$m_i = \frac{m_{max}}{C+2} i \qquad \text{(D.13)}$$

In the frequency domain, the ith triangular filter is defined as:

$$y_i(f) = \begin{cases} b_i(f - f_i) & \text{, if } f_{i-1} < f \leq f_i \\ -b_{i+1}(f - f_{i+1}) & \text{, if } f_i < f \leq f_{i+1} \\ 0 & \text{, otherwise} \end{cases} \qquad \text{(D.14)}$$

where f is the frequency variable, $y_i \in [0,1]$ depends on f, f_i is the middle of the ith triangular filter defined by substituting Equation (D.13) into Equation (D.12), and the slope $b_i = \frac{1}{f_i - f_{i-1}}$.

The power of the ith filtered window is a discrete sum of weighted (filtered) frequency bins:

$$P_i = \sum_{k=1}^{C} y_i(f_k) p_k \qquad \text{(D.15)}$$

Next, we scale these powers by performing a log tansform:

$$P_i \rightarrow \log(P_i) \qquad \text{(D.16)}$$

Finally, we take the Discrete Cosine Transform (DCT) of all filtered and log powers to obtain the MFCCs:

$$\sum_{i=1}^{C} \log(P_i) \xrightarrow{\text{DCT}} \sum_{i=0}^{C-1} MFCC_i \tag{D.17}$$

where $MFCC_i$ represents the ith Mel-Frequency Cepstral Coefficient, and $C = 12$ for use in Laκsa. The resultant cepstrum with MFCCs as coefficients characterize the spectral nature of the FFT spectrum.

D.3.2.11 SUMMARY OF INPUT FEATURES

Table D.2 summarizes the input features extracted for Sound inference.

Feature	Pretty Name	Icon	Conversion / Transformation	Units
Power	Volume	◢	$\text{Volume} = \max(P + 100, 0)$	0 to 100
Low-Energy Frame Rate [W]	Periods of Silence	🤐	$\text{PeriodsOfSilence} = 100 \times LEFR$	percent %
Bandwidth [P]	Pitch Range			Hz
Spectral Flux	Pitch Fluctuation			Hz/sec
Relative Spectral Entropy	Pitch Purity			bits
Zero Crossing Rate	Remaining Factors			
Spectral Rolloff [P]				
Spectral Centroid [P]				
Normalized Weighted Phase Detection				
MFCCs 0 to 11				

Table D.2. Summary of input features for Sound inference with simplified names, transformations, and units used in explanations to end-users. [W] Low-energy frame rate is a window feature where we only measure counts; all other features are time frame features where we measure both Means and Standard Deviations across time frames within the window. [P] Spectral Centroid, Bandwidth, and Spectral Rolloff were aggregated in Laκsa with the pan-flute visualization (see Figure 7.3c), but were separated for Laκsa2.

D.3.3 MACHINE LEARNING TO INFER SOUND ACTIVITY

Feature	Measure	Speech	Music	Ambient
Power	Mean	High	High	Low
	SD			
Low-Energy Frame Rate	Count	High	Low	Low
Zero Crossing Rate	Mean			
	SD	High	Usually Low	Low
Spectral Flux	Mean	High	Usually Low	
	SD			
Spectral Rolloff	Mean	Low	High	Low
	SD			
Spectral Centroid	Mean	Low	High	Low
	SD			
Bandwidth	Mean	Low	High	Low
	SD			
Normalized Weighted Phase Detection	Mean	High	Low	Low
	SD			
Relative Spectral Entropy	Mean	High	Low	Low
	SD			
Mel-Frequency Cepstral Coefficients (0 to 11)	Mean			
	SD			

Table D.3. Summary of input features for Sound inference and expected values for different class values (Speech, Music, and Ambient Noise).

Currently, Laκsa and Laκsa2 use the naïve Bayes classifier to infer three types of sound:

- Talking
- Music
- Ambient Noise

Note that the classifier was not trained to be particularly accurate, since the scenarios used in the user studies (Chapters 7 and 9) tested failure cases.

Why and Why Not explanations of Sound inference are generated using `NaiveBayesExplainer` and presented as Weights of Evidence bar chart visualizations. Inputs explanations are presented by describing the values of the input features with their pretty names and units. Description explanations are used to describe and explain the concepts relevant for each input feature.

D.4 MOTION ACTIVITY

Laκsa infers motion activity by

1. Sampling from the tri-axial accelerometer signals,
2. Extracting features from the signal, and
3. Applying machine learning to distinguish between different types of motion

We perform sampling and feature extraction using the methods similar to similar to [Bao and Intille, 2004; Lester, Choudhury, and Borriello, 2006] with a focus on using intelligible features. However, the inference uses the simpler J48 Decision Tree classifier which is explainable with the Intelligibility Toolkit.

D.4.1 MOTION SAMPLING FROM ACCELEROMETER

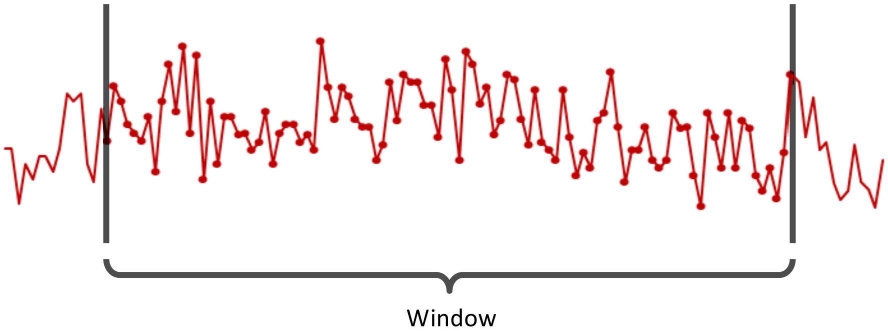


Figure D.5. Sampling of an accelerometer signal partitioned into a Window for sampling.

D.4.1.1 TIME SERIES SAMPLING

Laκsa samples three signals of the tri-axial accelerometer at a rate of 20 samples/sec in time windows of 6.4 seconds to measure 128 samples (see Figure D.5). We denote the tth sample in the window as s_t for a generic signal. We measure three signals representing linear acceleration ($\ddot{x}$, $\ddot{y}$, $\ddot{z}$) and calculate three other signals representing angular orientation (ψ, φ, θ). Independent of CPU time for feature extraction and classification, Laκsa requires at least 6.4 seconds to sample the accelerometer. Table D.4 summarizes the sampling characteristics for audio.

D.4.1.2 FAST FOURIER TRANSFORM TO OBTAIN FREQUENCY SPECTRUM

Similarly to how we pre-processed the audio signal for Sound inference, we perform a Fast Fourier Transform (FFT) to obtain spectral information of the signal, s:

$$\sum_{t=1}^{N} s_t \xrightarrow{\text{FFT}} \sum_{k=1}^{M} S_k \tag{D.18}$$

where $S_k = a_{sk} + ib_{sk}$ is a complex number representing the kth bin in the frequency series, $\mathrm{Re}(S_k) = a_{sk}$ is its real component and $\mathrm{Im}(S_k) = b_{sk}$ is its imaginary component. For Motion, we compute the spectra for six signals: linear acceleration ($\ddot{x}$, $\ddot{y}$, $\ddot{z}$) and angular orientation (ψ, φ, θ).

By considering the FFT output of complex numbers, we can calculate the amplitude of each frequency bin:

$$\text{Frequency amplitude: } A_{sk} = |S_k| = \sqrt{a_{sk}^2 + b_{sk}^2}$$

We use the frequency spectrum primarily to model motion instead of stationary orientation, so we normalize the DC component ($k = 1$) to 0, *i.e.*, $A_{s1} = 0$.

The characteristics of the frequency spectrum is summarized in Table D.4.

Domain	Characteristic	Formula	Value
Time Series	Sampling rate	S	20 samples/sec
	Sample duration (time frame length)	T	6.4 sec
	Number of samples	$N = S \times T$	128 samples
Frequency	Number of bins	$M = N/2$	64 bins
	Maximum resolvable frequency	$F_{max} \equiv F_{Nyquist} = S/2$	10 Hz
	Frequency Resolution	$dF = S/N \equiv F_{max}/M$	0.15625 Hz

Table D.4. Sampling characteristics from each accelerometer signal.

D.4.2 FEATURES EXTRACTED FROM THE ACCELEROMETER

Laκsa extracts several features from the tri-axial accelerometer signal using statistical measures from the time series data and the FFT spectrum.

D.4.2.1 3D LINEAR ACCELERATIONS

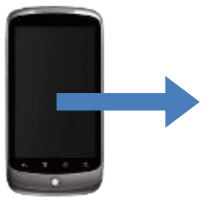
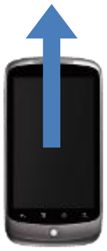
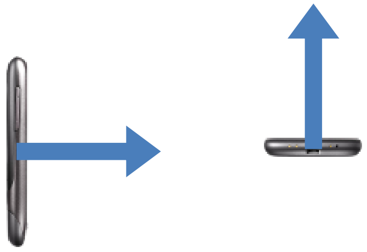
The raw signal value, at time t, from the phone's accelerometer represents the acceleration in three dimensions:

$$\vec{\ddot{r}}_t = \begin{pmatrix} \ddot{x}_t \\ \ddot{y}_t \\ \ddot{z}_t \end{pmatrix} \tag{D.19}$$

This constitutes the basis for many of the subsequent features for recognizing motion. For each linear signal, we measure their means and standard deviations over the sample time:

$$\begin{aligned} \mu_{\ddot{x}} &= \frac{1}{N}\sum\nolimits_{t=1}^{N} \ddot{x}_t & \sigma_{\ddot{x}} &= \frac{1}{N}\sqrt{\sum\nolimits_{t=1}^{N} (\ddot{x}_t - \mu_{\ddot{x}})^2} \\ \mu_{\ddot{y}} &= \frac{1}{N}\sum\nolimits_{t=1}^{N} \ddot{y}_t & \sigma_{\ddot{y}} &= \frac{1}{N}\sqrt{\sum\nolimits_{t=1}^{N} \left(\ddot{y}_t - \mu_{\ddot{y}}\right)^2} \\ \mu_{\ddot{z}} &= \frac{1}{N}\sum\nolimits_{t=1}^{N} \ddot{z}_t & \sigma_{\ddot{z}} &= \frac{1}{N}\sqrt{\sum\nolimits_{t=1}^{N} (\ddot{z}_t - \mu_{\ddot{z}})^2} \end{aligned} \tag{D.20}$$

Diagrammatically, with respect to the phone, its x, y, and z axes are:

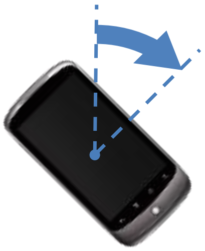
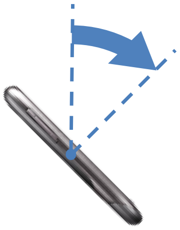
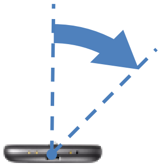
x Axis	*y* Axis	*z* Axis
		
Direction of positive x acceleration	Direction of positive y acceleration	Direction of positive z acceleration (two views)

D.4.2.2 3D ANGULAR ORIENTATION

The accelerations can also be considered in angular frames instead of linear frames. If we assume no or minimal actual motion, then the angular information from the accelerations can be interpreted as the static orientation of the phone:

$$\begin{aligned} Yaw &= \psi_t = \operatorname{atan}(\ddot{y}_t/\ddot{x}_t) \\ Roll &= \varphi_t = \operatorname{atan}(\ddot{z}_t/\ddot{y}_t) \\ Pitch &= \theta_t = \operatorname{atan}(\ddot{x}_t/\ddot{z}_t) \end{aligned} \tag{D.21}$$

Diagrammatically, with respect to the phone, its yaw, roll, and pitch are:

Yaw	Roll	Pitch
		
Rotation in xy plane, about z axis	Rotation in yz plane, about x axis	Rotation in zx plane, about y axis

For each angular orientation, we measure their means and standard deviations in the window time:

$$\begin{aligned} \mu_\psi &= \frac{1}{N}\sum_{t=1}^{N}\psi_t \quad & \sigma_\psi &= \frac{1}{N}\sqrt{\sum_{t=1}^{N}(\psi_t-\mu_\psi)^2} \\ \mu_\varphi &= \frac{1}{N}\sum_{t=1}^{N}\varphi_t \quad & \sigma_\varphi &= \frac{1}{N}\sqrt{\sum_{t=1}^{N}(\varphi_t-\mu_\varphi)^2} \\ \mu_\theta &= \frac{1}{N}\sum_{t=1}^{N}\theta_t \quad & \sigma_\theta &= \frac{1}{N}\sqrt{\sum_{t=1}^{N}(\theta_t-\mu_\theta)^2} \end{aligned} \tag{D.22}$$

D.4.2.3 Correlations between Linear Accelerations and Angular Orientations

We compute the correlations between linear accelerations and between angular components:

$$\begin{aligned} \rho_{\ddot{x}\ddot{y}} &= \frac{\sum_{t=1}^{N}\ddot{x}_t\ddot{y}_t - N\mu_{\ddot{x}}\mu_{\ddot{y}}}{(N-1)\sigma_{\ddot{x}}\sigma_{\ddot{y}}} \qquad & \rho_{\psi\varphi} &= \frac{\sum_{t=1}^{N}\psi_t\varphi_t - N\mu_\psi\mu_\varphi}{(N-1)\sigma_\psi\sigma_\varphi} \\ \rho_{\ddot{y}\ddot{z}} &= \frac{\sum_{t=1}^{N}\ddot{y}_t\ddot{z}_t - N\mu_{\ddot{y}}\mu_{\ddot{z}}}{(N-1)\sigma_{\ddot{y}}\sigma_{\ddot{z}}} \qquad & \rho_{\varphi\theta} &= \frac{\sum_{t=1}^{N}\varphi_t\theta_t - N\mu_\varphi\mu_\theta}{(N-1)\sigma_\varphi\sigma_\theta} \\ \rho_{\ddot{z}\ddot{x}} &= \frac{\sum_{t=1}^{N}\ddot{z}_t\ddot{x}_t - N\mu_{\ddot{z}}\mu_{\ddot{x}}}{(N-1)\sigma_{\ddot{z}}\sigma_{\ddot{x}}} \qquad & \rho_{\theta\psi} &= \frac{\sum_{t=1}^{N}\theta_t\psi_t - N\mu_\theta\mu_\psi}{(N-1)\sigma_\theta\sigma_\psi} \end{aligned} \tag{D.23}$$

D.4.2.4 DEVIATION OF ACCELERATION FROM 1G

We define a variable to potentially indicate whether the phone is stationary or in motion:

$$\Delta\ddot{r}_t = \ddot{r}_t - \ddot{r}_g \qquad \text{(D.24)}$$

where $\ddot{r}_t = \left|\vec{\ddot{r}}_t\right| = \sqrt{\ddot{x}_t^2 + \ddot{y}_t^2 + \ddot{z}_t^2}$ is the magnitude of the acceleration vector and $\ddot{r}_g = 9.81\text{m/s}^2$ is the constant of 1 G acceleration. If there is absolutely no motion in the phone, *e.g.*, it is resting flat on a table, then $\ddot{r}_t = 1$ G and $\Delta\ddot{r}_t = 0$, *i.e.*,

$$\begin{array}{ll} \Delta\ddot{r}_t = 0 & \text{, likely stationary} \\ \Delta\ddot{r}_t \neq 0 & \text{, likely in motion} \end{array}$$

Note that there may be certain cases when the phone is in motion and $\Delta\ddot{r}_t = 0$.

D.4.2.5 POWER (ENERGY PER UNIT TIME)

Taking the square of amplitudes in the frequency (FFT) spectrum, we can power spectral density (PSD) of the signal, s:

$$E_s = \langle s, s \rangle = \sum_{k=1}^{M} A_{sk}^2 \qquad \text{(D.25)}$$

where A_{sk}^2 is the power of the kth frequency bin. The area under the PSD curve represents the energy in the signal over the sampling time (window length). To get the average power over time, we divide by the window length (as was done in [Bao and Intille, 2004]). The average, normalized power for each signal is:

$$P_s = \frac{1}{T}\sum_{k=1}^{M} A_{sk}^2 \qquad \text{(D.26)}$$

computed for six signals: linear acceleration ($\ddot{x}$, $\ddot{y}$, $\ddot{z}$) and angular orientation (ψ, φ, θ).

D.4.2.6 MODAL AND MEAN FREQUENCIES

For each signal, we compute the mean frequencies:

$$\bar{f}_s = \frac{1}{M}\sum_{k=1}^{M} A_s \qquad \text{(D.27)}$$

and modal frequencies:

$$f_s^{mode} = f_{s\kappa} = \underset{\kappa<M}{\operatorname{argmax}} \left[\left(\sum_{k=1}^{\kappa} A_s \right) \geq \bar{f}_s \right] \tag{D.28}$$

computed for six signals: linear acceleration ($\ddot{x}$, $\ddot{y}$, $\ddot{z}$) and angular orientation (ψ, φ, θ).

D.4.2.7 Spectral Entropy

Spectral Entropy (SE) indicates the shape of the frequency spectrum:

$$SE_s = -\sum_{k=1}^{M} p_{sk} \log(p_{sk}) \tag{D.29}$$

computed for six signals: linear acceleration ($\ddot{x}$, $\ddot{y}$, $\ddot{z}$) and angular orientation (ψ, φ, θ). Spectral Entropy may indicate whether a particular frequency has high amplitude while most others have low amplitude (*e.g.*, when cycling at a constant speed).

D.4.2.8 Summary of Input Features

Table D.5 summarizes the input features extracted for Motion inference.

Feature	**Pretty Name**		**Icon**	**Conversion**	**Units**
$P_{\ddot{y}}$ power	Movement Vigorousness (Up/Down)				Watts
P_θ power	Movement Vigorousness (Rotation)				Watts
$\Delta\ddot{r}_t$	Amount of Movement (Sideways)			$Inches = m/39.37$	Inches ″
θ pitch (Std. Dev.)	Amount of Movement (Rotation)		Δ_{zx}	$\angle = 180° \cdot Radians/\pi$	Degrees °
$\ddot{x}$ acceleration (Mean)	x-Force		x	$Force = Accel/9.81$	G (=9.81m/s^2)
$\ddot{y}$ acceleration (Mean)	y-Force		y		
$\ddot{z}$ acceleration (Mean)	z-Force		z		
ψ yaw (Mean)	$\angle xy$ Rotation	Also shown as a physical diagram of the phone's orientation (see Figure 7.3b).	$\angle_{xy}$	$\angle = 180° \cdot \frac{Radians}{\pi}$	Degrees °
φ roll (Mean)	$\angle yz$ Rotation		$\angle_{yz}$		
θ pitch (Mean)	$\angle zx$ Rotation		$\angle_{zx}$		
Other features were not shown in explanations to users.					

Table D.5. Summary of input features in Laksa2 for Motion inference with simplified names, transformations, and units used in explanations to end-users.

D.4.3 MACHINE LEARNING TO INFER MOTION ACTIVITY

Currently, Laκsa and Laκsa2 use the J48 Decision Tree classifier to infer seven types of motion:

- Sitting
- Standing
- Walking
- Running
- Cycling
- Holding
- 'Flat Surface' (*e.g.*, lying on a table)

Note that the classifier was not trained to be particularly accurate, since the scenarios used in the user studies (Chapters 7 and 9) tested failure cases.

Why and Why Not explanations of Motion inference are generated using `J48RuleTraceExplainer` and presented as inequalities (rule trace explanations). Inputs explanations are presented by describing the values of the input features with their pretty names and units. Description explanations are used to describe and explain the concepts relevant for each input feature.

E LAKSA EXPERIMENT MATERIALS

This appendix describes the experimental materials used in the user study described in Chapter 7.

E.1 PARTICIPANT REFERENCE SHEET

Cheat sheet that participants can carry around as they performed activities during scenarios in the experiment.

Personal Availability Rules

Availability	Location	Motion	Sound	Ringer	Schedule		Who's Enquiring
Unavailable	Office			Silent			*Anyone*
Unavailable	Office		Talking				*Anyone*
Unavailable					Work		Friends
Semi-Available	Library						*Anyone*
Semi-Available	Home						Coworkers
Semi-Available		Cycling					*Anyone*
Available	*All other cases*						

Explanation Types

1. **What** is the value of the aspect?
2. **When** did this aspect change to this value?
3. **Why** is this aspect the current value? (This refers to the logic or method that Laksa used)
4. **Why isn't** this aspect an alternative value?
5. **When would** (under what situations or circumstances) this aspect?
6. **What else** can this aspect be? (What other values can it take?)
7. **What details** affect this aspect? (Factors, related details, etc)
8. **What if** the conditions are different, what would this aspect be? (Requires your manipulation)
9. ⓘ More information (e.g. meaning of some terms, values).

F Uncertainty Study Materials

This appendix describes experiment design notes and materials used in the user study described in Chapter 8.

F.1 SCENARIOS

F.1.1 SCENARIO DISTRIBUTION BY THEME

Distribution of scenarios for learning phase (10 scenarios), and testing phase (3 scenarios).

F.1.1.1 LEARNING PHASE

S#		Theme / Topic	LocateMe (location-aware)	HearMe (sound-aware)
1	1a	Social Awareness	Morning, want to check if friend is home, to drop by (he isn't home): may not infer as home	Morning, want to call friend: may be inferred as talking when actually silent
2	2a	Awareness / Interruptibility	In office: coworker wants to check if you are in office yet, to talk to you	Talking to coworker, may be inferred as ambience: get call, or get notification msg of suppressed call.
3	3a	(Pre-set) Service A	In Meeting Room A: but may mis-infer that you're not at venue -> send reminder	After talking, want to review audio of conversation: may not have recorded since inferred as Music.
4	4a	Recommender Service B	In corridor, want to print document from phone to nearest printer: may print elsewhere	Detected silence for a long time; recommends music
5	5a	Self check	Visit coffee shop: make sure location noted	Visit coffee shop: make sure sound noted as ambient, not talking
6	2b	Awareness / Interruptibility	In washroom: coworker may wonder why you appear to be in his office	Talking to *coworker* at coffee shop, may be inferred as music: get call, or get notification msg of suppressed call.
7	4b	Recommender Service B	In *another* corridor, want to print document from phone to nearest printer: may print elsewhere	Detected silence for a long time; recommends music
8	5b	Self check	Walking outside to lunch, self-check: location may be still in office	Testing/checking app, listening to (vocal) music: may be wrongly inferred as talking
9	3b	(Pre-set) Service A	In Meeting Room B: but may mis-infer that you're not at venue -> send reminder	After talking to team of coworkers, want to review audio of conversation: may not have recorded since inferred as Music.
10	1b	Social Awareness	Evening, want to check if friend is home, to drop by (he is home): may not infer as home	Evening, want to call friend: may be inferred as talking when actually music

F.1.1.2 Testing Phase

S#		Theme / Topic	LocateMe (location-aware)	HearMe (sound-aware)
11	2c	No theme; Just show ground truth	Just show Map.	Just play Audio clip.
12	5c			
13	4c			

F.1.2 Scenario Distribution by Application Behavior

For each Certainty condition, scenarios were randomly selected where the application behaved appropriately (✓) or inappropriately (×). Certainty values also have a "fudge" factor to introduce noise to their values so that they appear random to participants and less predictable.

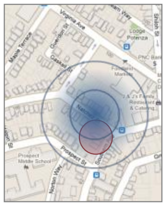
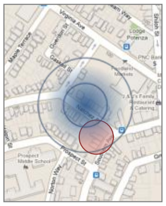
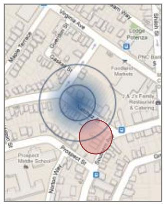
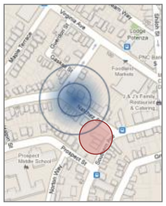
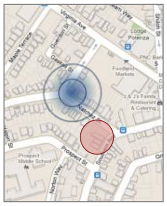
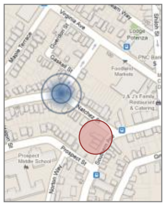
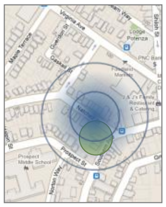
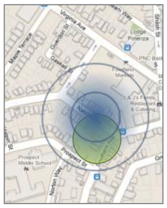
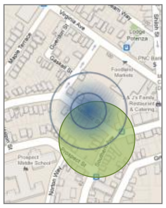
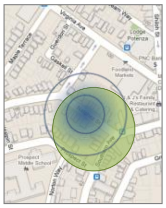
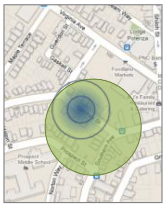
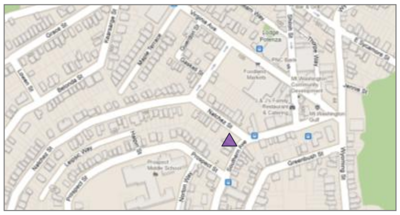
S#		Both LocateMe and HearMe (%)											
		50		60		70		80		90		100	
1	1a	48	×	60	×	72	×	79	×	88	✓	100	✓
2	2a	51	✓	63	✓	72	✓	80	✓	89	✓	98	✓
3	3a	48	✓	59	✓	67	✓	80	✓	90	✓	99	✓
4	4a	51	×	60	×	69	×	79	✓	92	✓	99	✓
5	5a	50	✓	58	✓	69	✓	82	✓	90	✓	100	✓
6	2b	52	×	62	×	70	×	79	×	89	×	100	✓
7	4b	51	✓	61	✓	72	✓	81	✓	88	✓	98	✓
8	5b	49	×	57	×	73	✓	82	✓	91	✓	99	✓
9	3b	49	✓	60	✓	68	✓	80	✓	91	✓	98	✓
10	1b	51	×	60	✓	68	✓	78	✓	92	✓	99	✓

F.2 SCENARIO SCRIPTS

We provide scripts and explanation visualizations for both applications LocateMe and HearMe for all 10 scenarios.These are shown in sections:

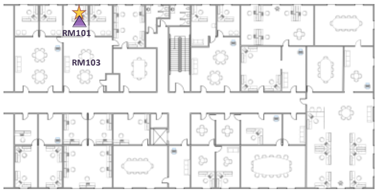
1. Scenario introduction
2. Ground truth description
3. Script and diagram for correct behavior (for different Certainty conditions)
4. Script and diagram for wrong behavior (for different Certainty conditions)

F.2.1 LOCATEME SCENARIOS

L1. Morning, want to check if friend is home, to drop by (he isn't home): may not infer as home					
Before you leave for work, you want to check if your friend, John, is still at home so that you can pick up a toolbox you had previously lent him.					
You look at LocateMe to see his location:					
Correct Application Response					
John is at Other Place					
50%	**60%**	**70%**	**80%**	**90%**	**100%**
					
It tells you John has already left his home. You decide to try again later in the day.					
Wrong Application Response					
John is at Home					
50%	**60%**	**70%**	**80%**	**90%**	**100%**
					N.A.
It tells you John is still at home. You call him, and discover he has already left home and is driving Natchez St. towards Gaskell St.					
Ground Truth					
his house is marked with a purple triangle on the following map).					

L2. In office: coworker wants to check if you are in office yet, to talk to you

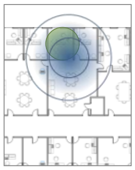
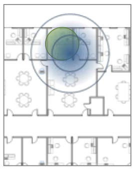
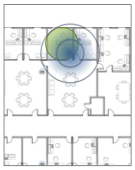
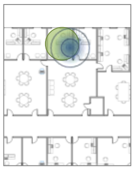
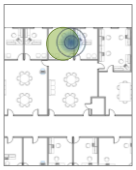
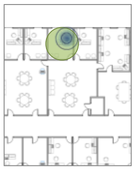
You are in your office getting some work done.
The star denotes where you actually are at for this scenario.
The triangle denotes where your office (RM101) is.



Correct Application Response

Your coworker, Michelle, visits your office and remarks that it is good that you are in early preparing for the important meeting later.
You comment that she could check your location with LocateMe.
You check your status and see:

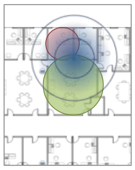
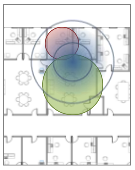
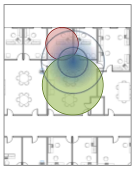
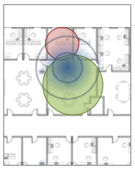
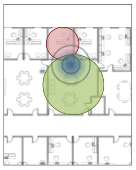
You are at **Your Office (RM101)**

50%	60%	70%	80%	90%	100%
					

Wrong Application Response

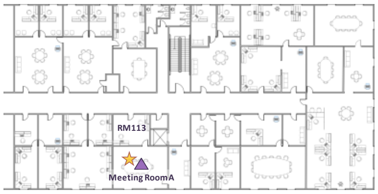
You get a call from your coworker, Michelle, who anxiously asked you whether you were in your office yet, preparing for the important meeting in a few minutes.
You check your status and see:

You are in **RM103**

50%	60%	70%	80%	90%	100%
					N.A.

L3. In meeting room A: but may misrecognize that you're not at venue -> send reminder

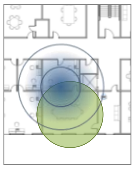
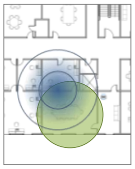
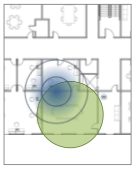
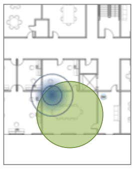
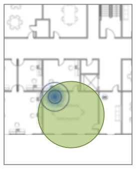
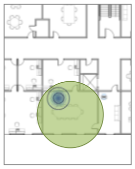
You arrive in Meeting Room A five minutes early.
The star denotes where you actually are at for this scenario.
The triangle denotes where Meeting Room A is.



Correct Application Response

LocateMe beeps and loads a document containing the meeting agenda.
You check your status and see:

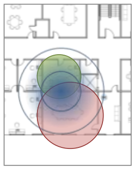
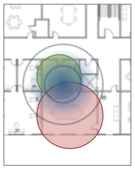
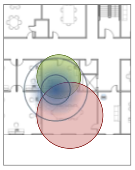
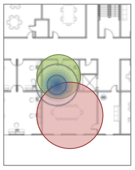
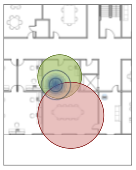
You are in **Meeting Room A**

50%	60%	70%	80%	90%	100%
					

Wrong Application Response

LocateMe beeps and sends you a reminder to get to your meeting in Meeting Room A:

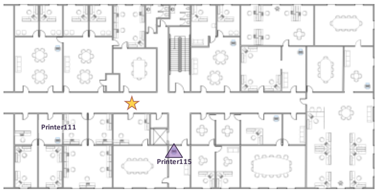
You are in **RM113**
Please go to your meeting in **Meeting Room A**

50%	60%	70%	80%	90%	100%
					N.A.

L4. In corridor, want to print email from phone to nearest printer: may print elsewhere

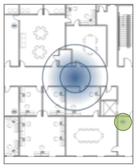
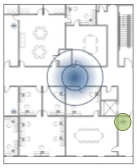
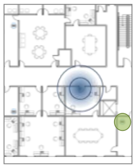
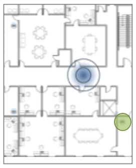
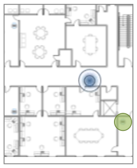
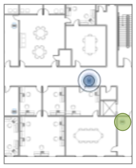
After the meeting, you are checking your email as you walk along a corridor.
You want to print out an email from your phone to the nearest printer.
The star denotes where you actually are for this scenario.

LocateMe has the function to do this automatically.
You select "Print to nearest printer."
You check which printer your email was printed to:



Correct Application Response

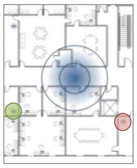
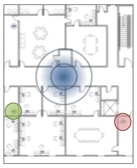
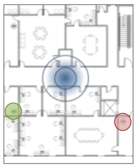
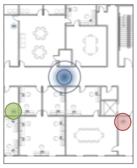
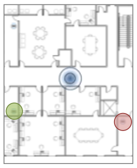
Nearest Printer is **Printer115**

50%	60%	70%	80%	90%	100%
					

Satisfied with the destination, you walk to the nearest printer, and collect your print out.

Wrong Application Response

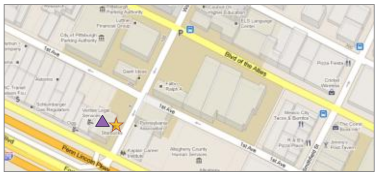
Nearest Printer is **Printer111**

50%	60%	70%	80%	90%	100%
					N.A.

This is not actually the nearest to you.

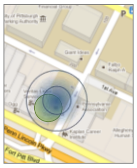
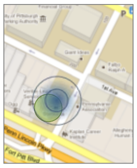
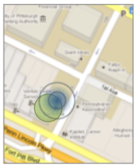
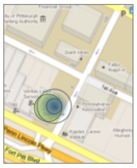
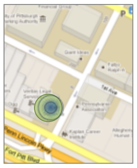
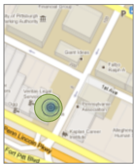
L5. Visit coffee shop: make sure location noted

Feeling sleepy, you visit a coffee shop near your office to get a cup of coffee.
Since LocateMe is a new application to you, you want to check whether it can properly detect you at the coffee shop.
You look at LocateMe and see:



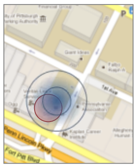
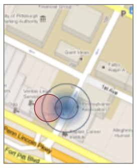
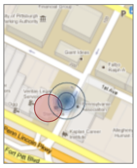
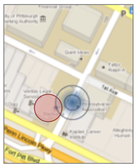
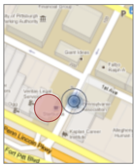
Correct Application Response

You are at **Starbucks Coffee**

50%	60%	70%	80%	90%	100%
					

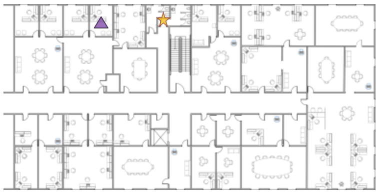
Wrong Application Response

You are at **Some Other Place**

50%	60%	70%	80%	90%	100%
					N.A.

L6 (L2b). In washroom: coworker may wonder why you appear to be in his office

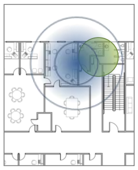
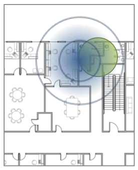
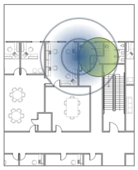
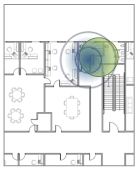
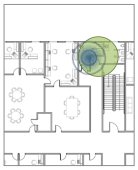
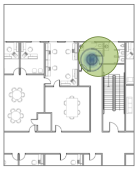
You are in the washroom taking care of some business, just before your meeting with your neighboring coworker, Damien.
The star denotes where you actually are at for this scenario.
The purple triangle denotes where Damien's office (RM102) is.



Correct Application Response

You receive a text message from your coworker, Damien, telling you he is waiting at his office, awaiting your arrival.
You check your status and see:

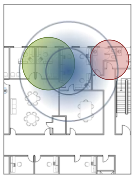
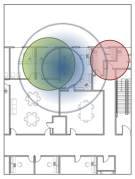
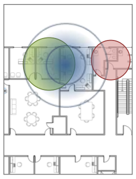
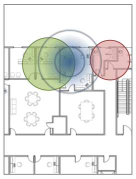
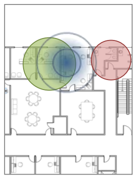
You are at **the Washroom**

50%	60%	70%	80%	90%	100%
					

Wrong Application Response

You receive a call from your neighboring coworker, Damien, asking you where you are since LocateMe told him that you are at his office but he obviously does not see you there.
You check your status and see:

You are at **Damien'sOffice (RM102)**

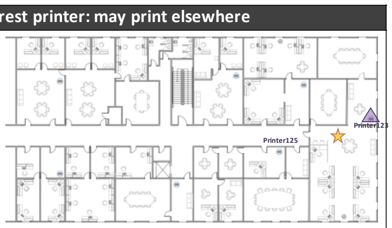
50%	60%	70%	80%	90%	100%
					N.A.

L7 (L4b). In another corridor, want to print document from phone to nearest printer: may print elsewhere

You want to print *another* email after asking a coworker to send it to you.
You select "Print to nearest printer."

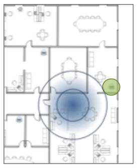
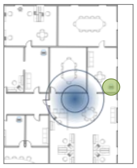
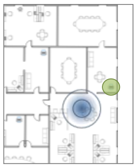
Note printers (with icons); nearest one is marked with triangle

You check which printer your email was printed to:



Correct Application Response

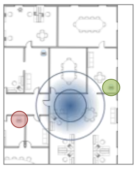
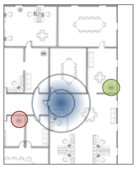
Nearest Printer is **Printer123**

50%	60%	70%	80%	90%	100%
					

Satisfied with the destination, you walk to the nearest printer, and collect your print out.

Wrong Application Response

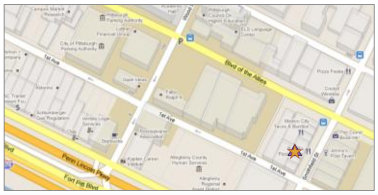
Nearest Printer is **Printer125**

50%	60%	70%	80%	90%	100%
					N.A.

This is not actually the nearest to you.

L8 (L5b). Walking outside to lunch, self-check: location may be still in office

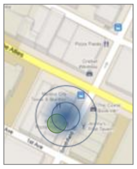
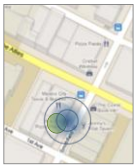
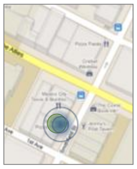
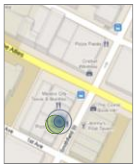
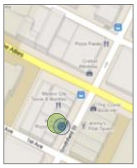
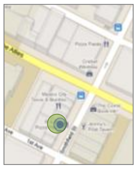
It is lunch time and you leave the office to go to a nearby pizza restaurant.
Since LocateMe is a new application to you, you want to check how it checks your location.
While having lunch at the restaurant, you look at LocateMe and see:



You look at LocateMe and see:

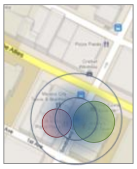
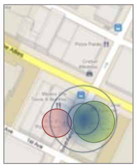
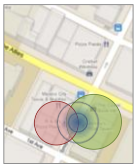
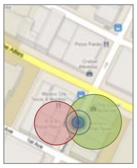
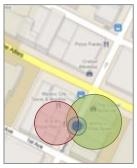
Correct Application Response

You are at **R & B's Pizza Place**

50%	60%	70%	80%	90%	100%
					

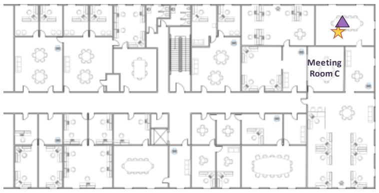
Wrong Application Response

You are at **Jimmy's Post Tavern**

50%	60%	70%	80%	90%	100%
					N.A.

L9 (L3b). In Meeting Room B: but may mis-infer that you're not at venue -> send reminder

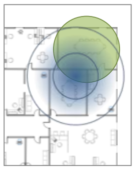
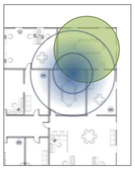
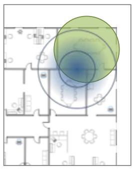
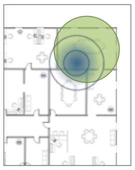
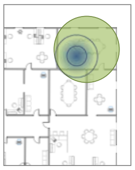
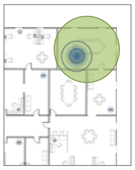
You arrive in Meeting Room B five minutes early for an afternoon meeting.
The star denotes where you actually are at for this scenario.



Correct Application Response

LocateMe beeps and loads a document containing the meeting agenda.
You check your status and see:

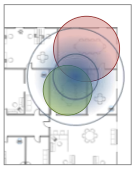
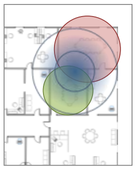
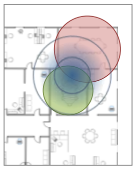
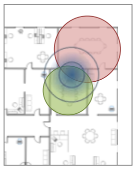
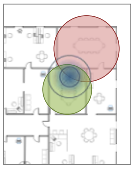
You are in **Meeting Room B**

50%	60%	70%	80%	90%	100%
					

Wrong Application Response

LocateMe beeps and sends you a reminder to get to your meeting in Meeting Room B:

You are in **Meeting Room C**
Please go to your meeting in **Meeting Room B**

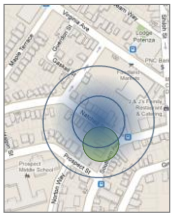
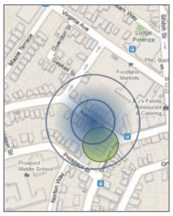
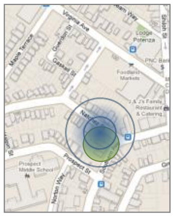
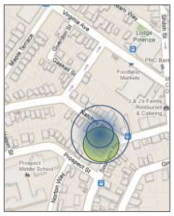
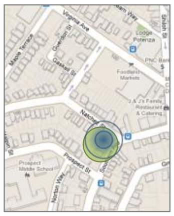
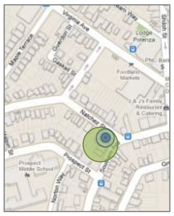
50%	60%	70%	80%	90%	100%
					N.A.

L10 (L1b). Evening, want to check if friend is home, to drop by (he is home): may not infer as home

Before you leave work, you want to check if your friend, John, has returned home so that you can pick up a toolbox you had previously lent him.
You look at LocateMe to see his location:

Correct Application Response

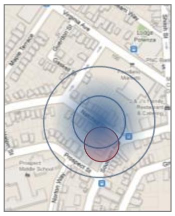
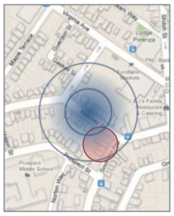
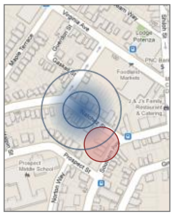
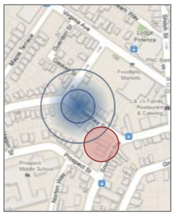
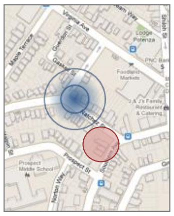
John is at **Home**

50%	60%	70%	80%	90%	100%
					

It tells you John is at home.
You call him, and let him know that you would be coming by his place.

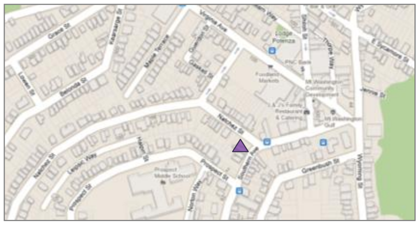
Wrong Application Response

John is at **Other Place**

50%	60%	70%	80%	90%	100%
					N.A.

It tells you John is not home.
You call him to ask him what time he would be home, but he tells you he already is.

Ground Truth

John is actually at home (his house is marked with a purple triangle on the following map).	

F.2.2 HearMe Scenarios

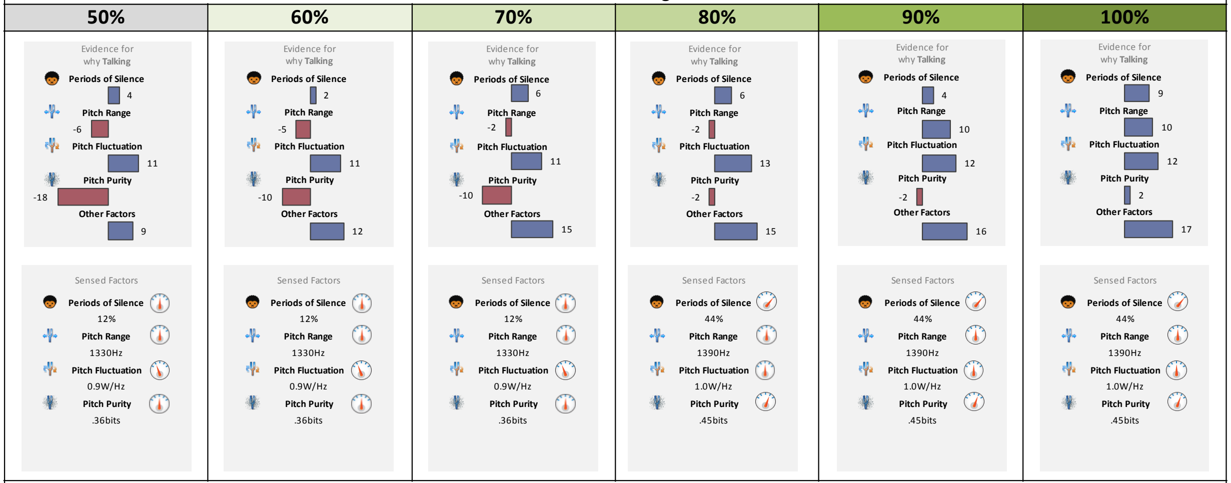
H1. Morning, want to call friend: may be inferred as talking when actually silent

Once you arrive at work in the morning, you remember you want to contact your friend, John.
You want to call John to ask to pick up, later in the evening, a toolbox you had previously lent him.
Before you do that, you want to check if he is available to be called.

You look at HearMe to see his status:

Correct Application Response

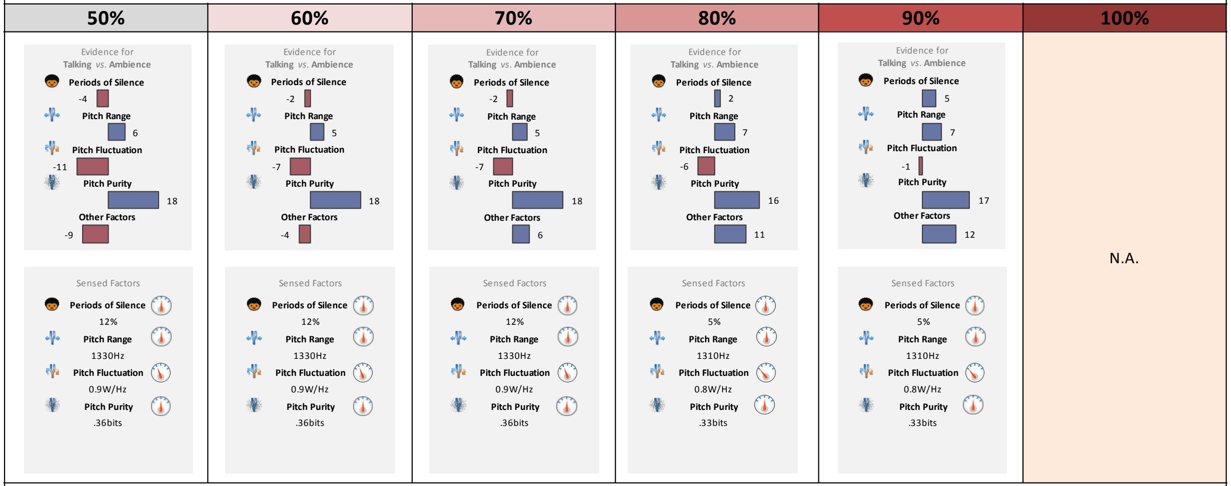
John is **Talking**



You send him a text message to ask him to call you when he is done talking.

Wrong Application Response

John is **Silent**



You call him, but he tells you he is busy talking to someone now, and asks you to call him back later in the day.

Ground Truth

<NO AUDIO>

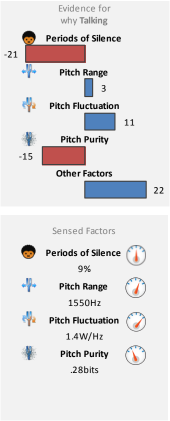
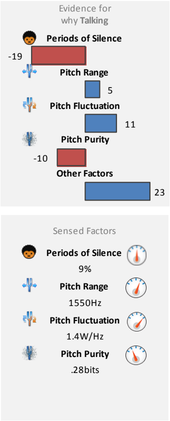
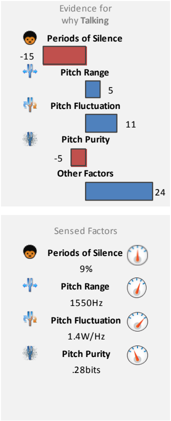
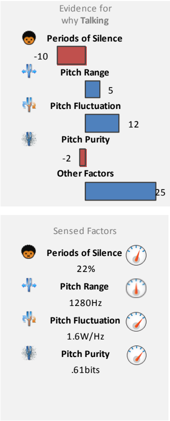
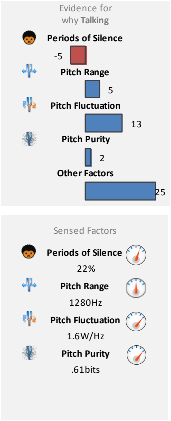
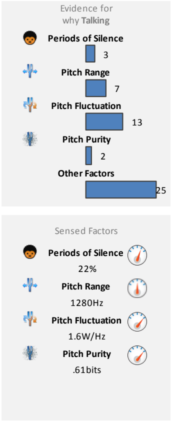
H2. Talking to coworker, may be inferred as ambience: get call, or get notification message of suppressed call.

Next, while at your office, your coworker, Michael, comes by to have a conversation with you about a project.	Here is an audio clip of what was actually heard at that time: **<AUDIO: h2-talking-work.mp3>**

Correct Application Response

You are not interrupted for 20 minutes, and when the conversation ends, you receive a notification message from HearMe:

You were **In a Conversation**
Jenny tried to call you 9 minutes ago

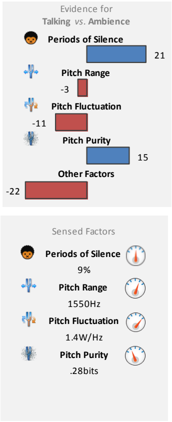
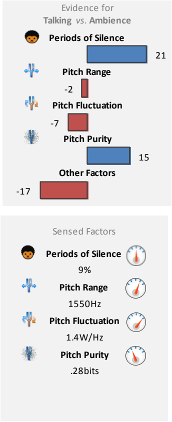
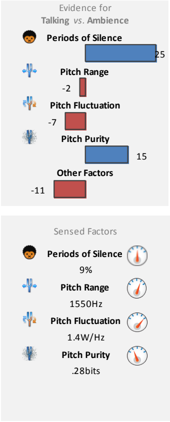
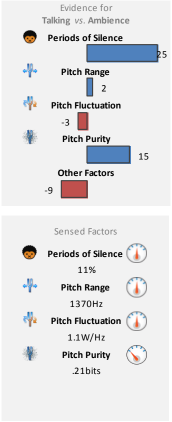
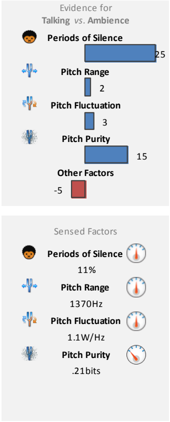
50%	60%	70%	80%	90%	100%
					

You see that your coworker, Jenny had tried to call you, but HearMe suppressed her call since it interpreted you as uninterruptible.

Wrong Application Response

After 11 minutes, midway through your conversation, you get an interruption from another coworker, Jenny, calling you on your phone.
You quickly ignore the call, and check HearMe's status of you:

Hearing **Ambience Noise**
Allowing call from Jenny

50%	60%	70%	80%	90%	100%
					N.A.

H3. After talking to boss, want to review audio of conversation: may not have recorded since inferred as Music.

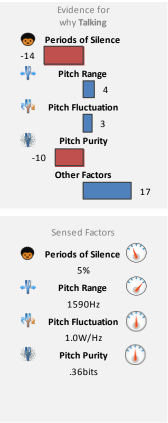
Later, you talk to your boss, and he made a comment. You want to review and save the recorded audio. HearMe has a feature to automatically detect speech, and record the last 30 minutes heard. It is supposed to only record if it heard speech, and not other types of sound, so that it does not record useless audio.	Here is an audio clip of what was actually heard at that time: **<AUDIO: h3-talking-work-boss.mp3>**

You review HearMe and see:

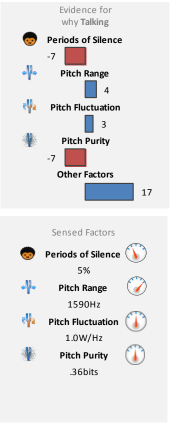
Correct Application Response

You were **In a Conversation**
Last recorded speech 1 minute ago

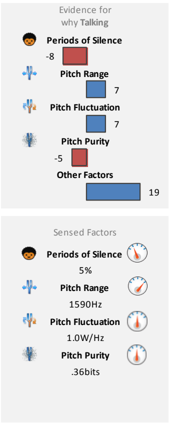
50%



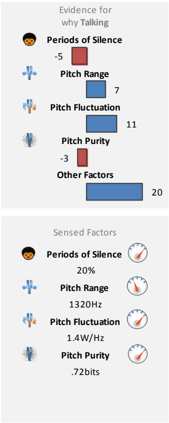
60%



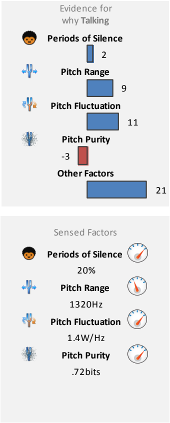
70%



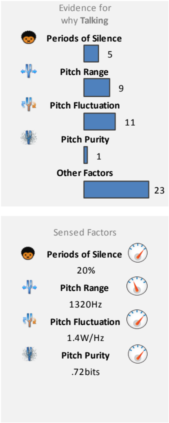
80%



90%



100%

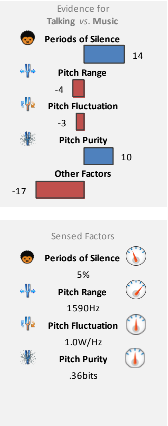


You then play back the rest of the speech, and save it.

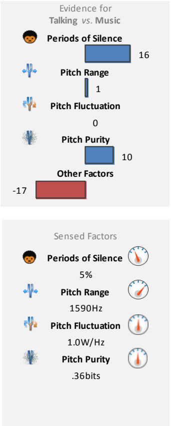
Wrong Application Response

You were **Listening to Music**
Last recorded speech 1 hour ago

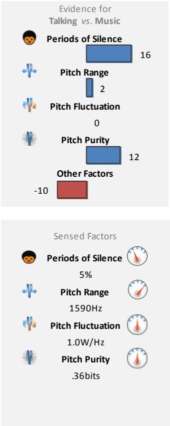
50%



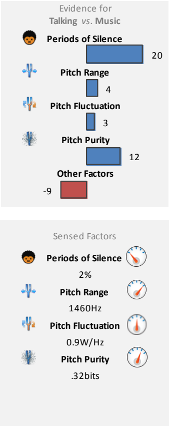
60%



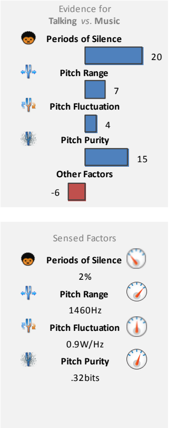
70%



80%



90%



100%

N.A.

You realize that the recent conversation was never recorded.

H4. Detected silence for a long time; recommends music

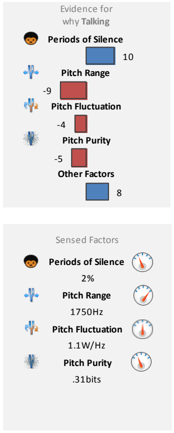
You are in your office working alone for some time. Normally you like to have music on as you work, but you have forgotten to turn it on.	Here is an audio clip of what was actually heard at that time: **<AUDIO: h4-office-ambient.mp3>**

Correct Application Response

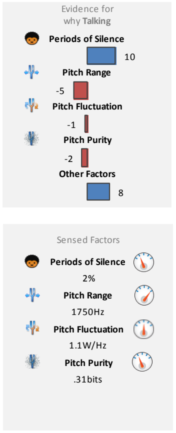
After a while, HearMe sends you a recommendation notification:

You have been **Silent**
Would you like to **Listen to Music**?

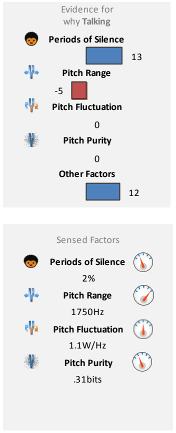
50%



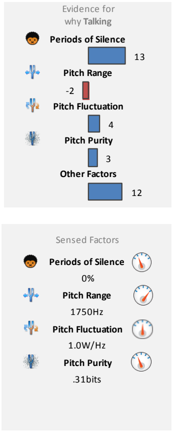
60%



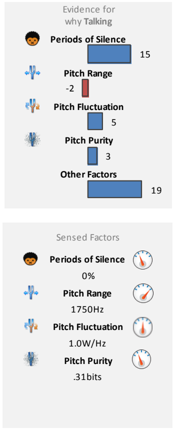
70%



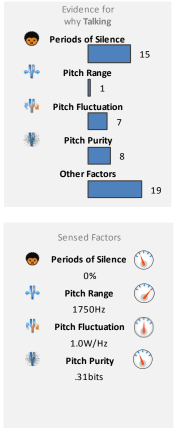
80%



90%



100%

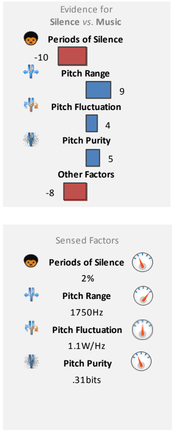


Wrong Application Response

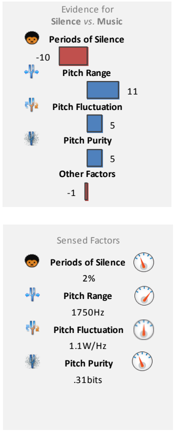
After a while, you realize that HearMe should have recommended you to play music, as is part of its functionality.
You examine HearMe:

You are **Listening to Music**

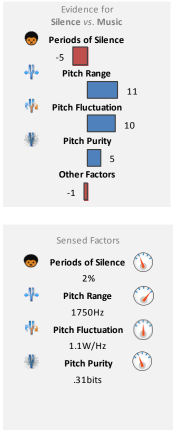
50%



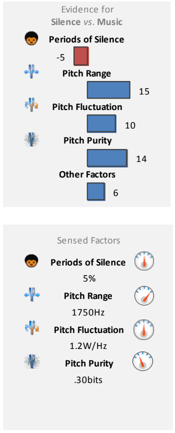
60%



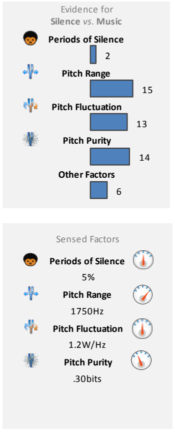
70%



80%



90%



100%

N.A.

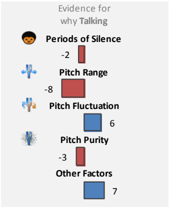
H5. Visit coffee shop: make sure sound noted as ambient, not talking

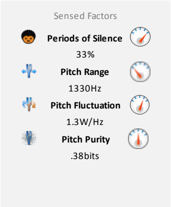
Feeling sleepy, you walk to the nearby coffee shop to get some coffee. Since HearMe is a new application to you, you want to check how it recognizes the noise in the coffee shop. You look at HearMe and see:	Here is an audio clip of what was actually heard at that time: **<AUDIO: h5-coffeeshop.mp3>**

Correct Application Response

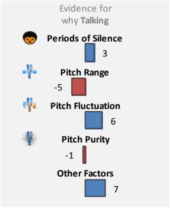
Hearing **Ambient Noise**

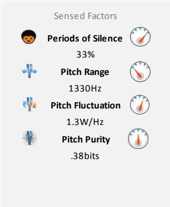
50%



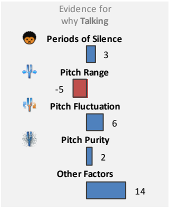


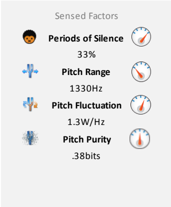
60%



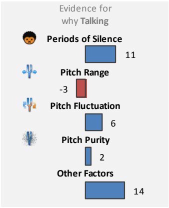


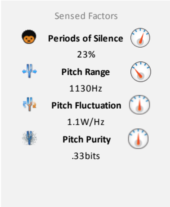
70%



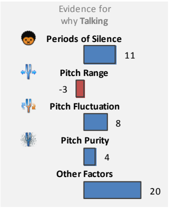


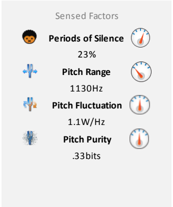
80%



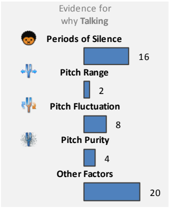


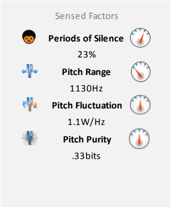
90%





100%

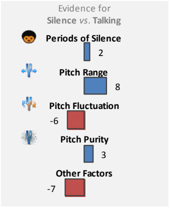


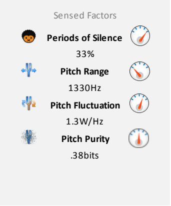


Wrong Application Response

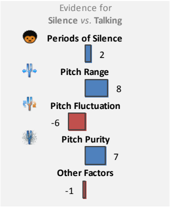
You are **In a Conversation**

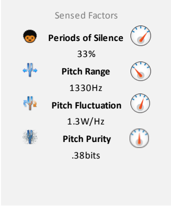
50%



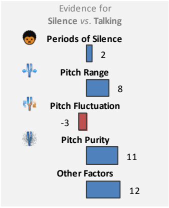


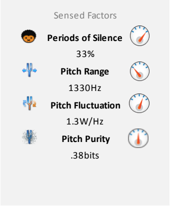
60%



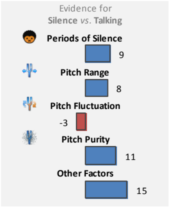


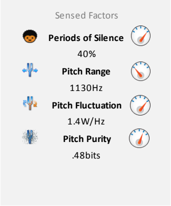
70%



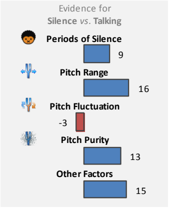


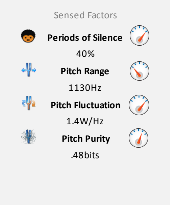
80%





90%





100%

N.A.

H6 (H2b). Talking to coworkers at coffee shop, may be inferred as music: get call, or get notification msg of suppressed call.

At the coffee shop, you find Michelle, a coworker, there, and have a chat with her.	Here is an audio clip of what was actually heard at that time: **<AUDIO: h6-talking-cafe.mp3>**

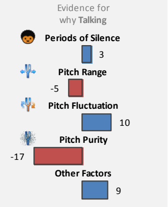
Correct Application Response

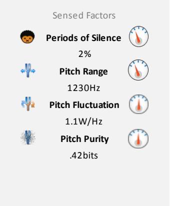
You are not interrupted for 12 minutes, and when the conversation ends, you receive a notification message from HearMe:

You were **In a Conversation**

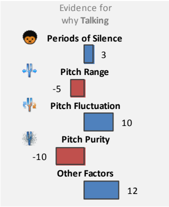
Cameron tried to call you 7 minutes ago

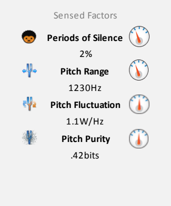
50%



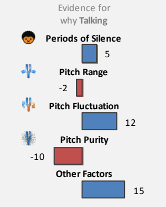


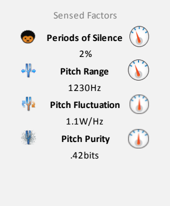
60%



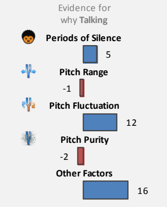


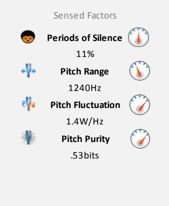
70%



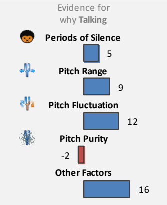


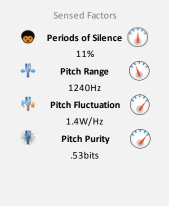
80%



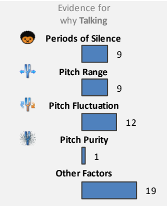


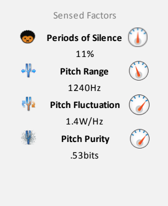
90%





100%





You see that your coworker, Cameron had tried to call you, but HearMe suppressed her call since it interpreted you as uninterruptible.

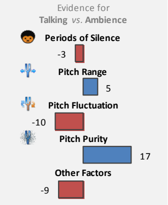
Wrong Application Response

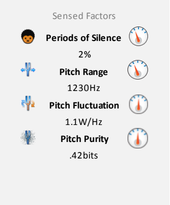
After 5 minutes, midway through your conversation, you get an interruption from another coworker, Cameron, calling you on your phone.
You quickly ignore the call, and check HearMe's status of you:

Hearing **Ambience Noise**

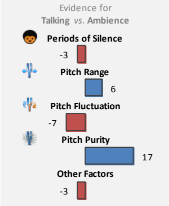
Allowing call from Cameron

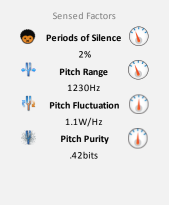
50%



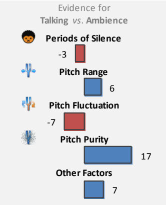


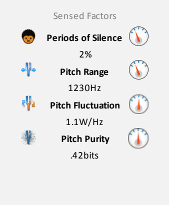
60%



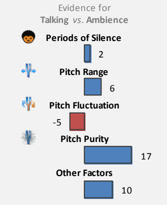


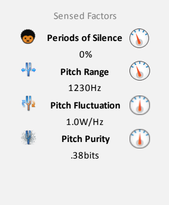
70%



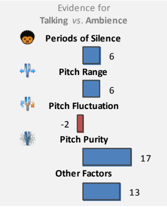


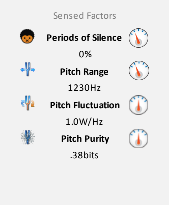
80%





90%





100%

N.A.

H7 (H4b). Detected silence for a long time; recommends music

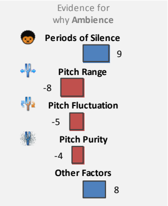
Once again, you are in your office working alone for some time, and you like to have music on as you work, but you have forgotten to turn it on.	Here is an audio clip of what was actually heard at that time: **<AUDIO: h7-office-noise.mp3>**

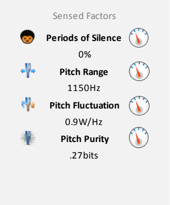
Correct Application Response

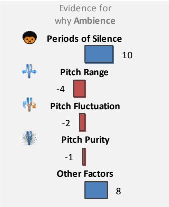
After a while, HearMe sends you a recommendation notification:

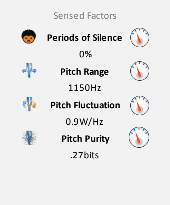
You have been **Silent**
Would you like to **Listen to Music**?

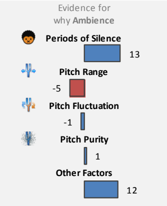
50%	60%	70%	80%	90%	100%

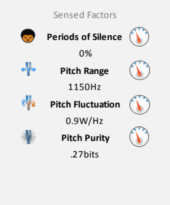


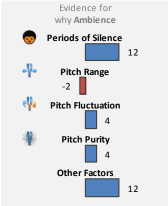


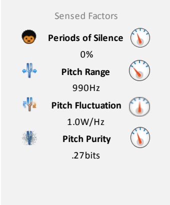


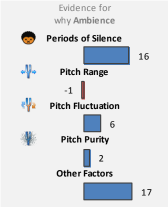


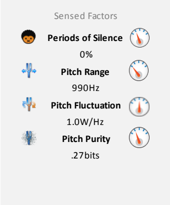


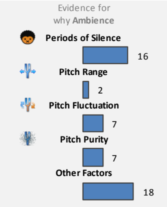


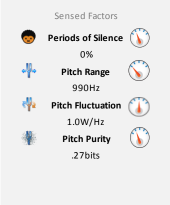










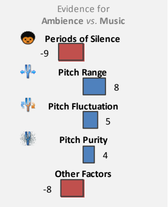


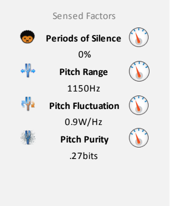
Wrong Application Response

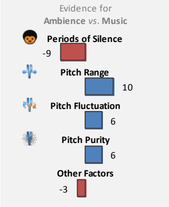
After a while, you realize that HearMe should have recommended you to play music, as is part of its functionality.
You examine HearMe:

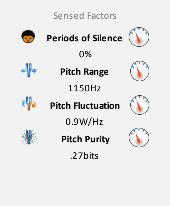
You are **Listening to Music**

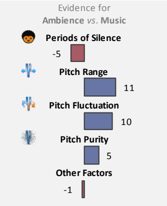
50%	60%	70%	80%	90%	100%

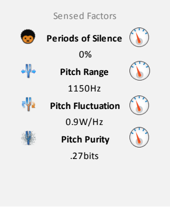


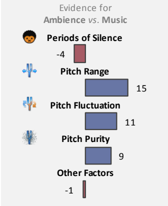


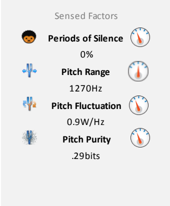


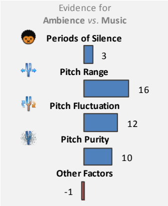


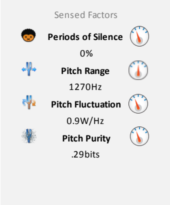












N.A.

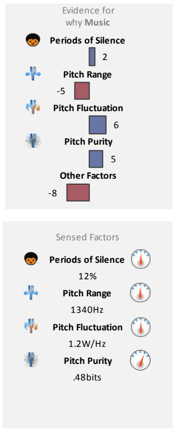
H8 (H5b). Testing/checking app, listening to (vocal) music: may be wrongly inferred as talking

You are now working and listening to some music by Josh Woodward. Since HearMe is a new application to you, you want to check how it recognizes the music you are playing. You look at HearMe and see:	Here is an audio clip of what was actually heard at that time: **<AUDIO: h8-music-vocal.mp3>**

Correct Application Response

You are **Listening to Music**

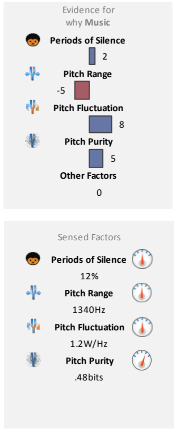
50%



Evidence for why **Music**
Periods of Silence 2
Pitch Range -5
Pitch Fluctuation 6
Pitch Purity 5
Other Factors -8

Sensed Factors
Periods of Silence 12%
Pitch Range 1340Hz
Pitch Fluctuation 1.2W/Hz
Pitch Purity .48bits

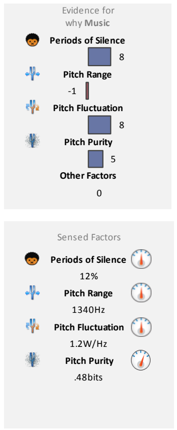
60%



Evidence for why **Music**
Periods of Silence 2
Pitch Range -5
Pitch Fluctuation 8
Pitch Purity 5
Other Factors 0

Sensed Factors
Periods of Silence 12%
Pitch Range 1340Hz
Pitch Fluctuation 1.2W/Hz
Pitch Purity .48bits

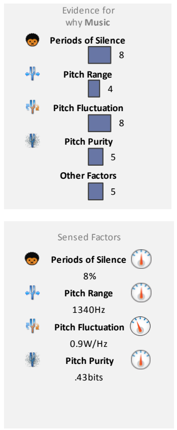
70%



Evidence for why **Music**
Periods of Silence 8
Pitch Range -1
Pitch Fluctuation 8
Pitch Purity 5
Other Factors 0

Sensed Factors
Periods of Silence 12%
Pitch Range 1340Hz
Pitch Fluctuation 1.2W/Hz
Pitch Purity .48bits

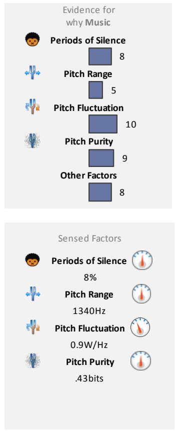
80%



Evidence for why **Music**
Periods of Silence 8
Pitch Range 4
Pitch Fluctuation 8
Pitch Purity 5
Other Factors 5

Sensed Factors
Periods of Silence 8%
Pitch Range 1340Hz
Pitch Fluctuation 0.9W/Hz
Pitch Purity .43bits

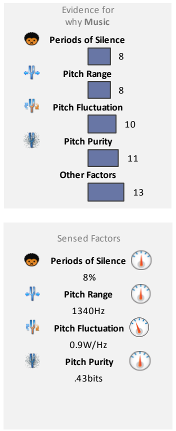
90%



Evidence for why **Music**
Periods of Silence 8
Pitch Range 5
Pitch Fluctuation 10
Pitch Purity 9
Other Factors 8

Sensed Factors
Periods of Silence 8%
Pitch Range 1340Hz
Pitch Fluctuation 0.9W/Hz
Pitch Purity .43bits

100%



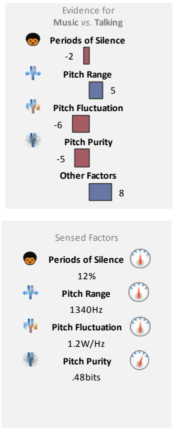
Evidence for why **Music**
Periods of Silence 8
Pitch Range 8
Pitch Fluctuation 10
Pitch Purity 11
Other Factors 13

Sensed Factors
Periods of Silence 8%
Pitch Range 1340Hz
Pitch Fluctuation 0.9W/Hz
Pitch Purity .43bits

Wrong Application Response

You are **In a Conversation**

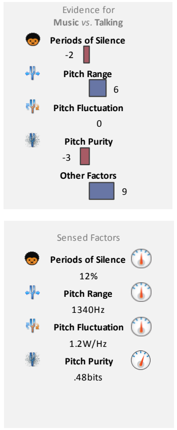
50%



Evidence for **Music** *vs.* **Talking**
Periods of Silence -2
Pitch Range 5
Pitch Fluctuation -6
Pitch Purity -5
Other Factors 8

Sensed Factors
Periods of Silence 12%
Pitch Range 1340Hz
Pitch Fluctuation 1.2W/Hz
Pitch Purity .48bits

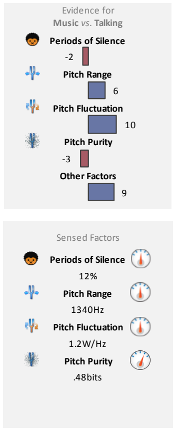
60%



Evidence for **Music** *vs.* **Talking**
Periods of Silence -2
Pitch Range 6
Pitch Fluctuation 0
Pitch Purity -3
Other Factors 9

Sensed Factors
Periods of Silence 12%
Pitch Range 1340Hz
Pitch Fluctuation 1.2W/Hz
Pitch Purity .48bits

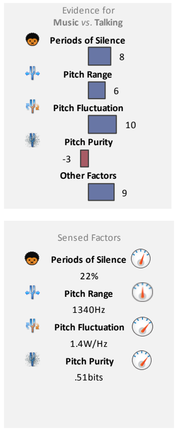
70%



Evidence for **Music** *vs.* **Talking**
Periods of Silence -2
Pitch Range 6
Pitch Fluctuation 10
Pitch Purity -3
Other Factors 9

Sensed Factors
Periods of Silence 12%
Pitch Range 1340Hz
Pitch Fluctuation 1.2W/Hz
Pitch Purity .48bits

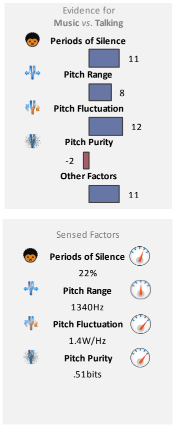
80%



Evidence for **Music** *vs.* **Talking**
Periods of Silence 8
Pitch Range 6
Pitch Fluctuation 10
Pitch Purity -3
Other Factors 9

Sensed Factors
Periods of Silence 22%
Pitch Range 1340Hz
Pitch Fluctuation 1.4W/Hz
Pitch Purity .51bits

90%



Evidence for **Music** *vs.* **Talking**
Periods of Silence 11
Pitch Range 8
Pitch Fluctuation 12
Pitch Purity -2
Other Factors 11

Sensed Factors
Periods of Silence 22%
Pitch Range 1340Hz
Pitch Fluctuation 1.4W/Hz
Pitch Purity .51bits

100%

N.A.

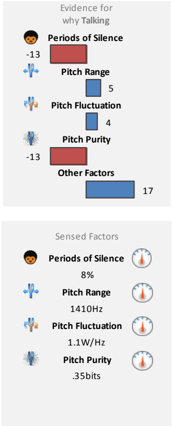
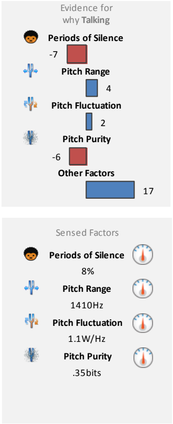
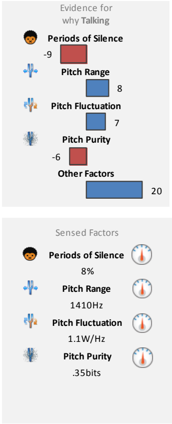
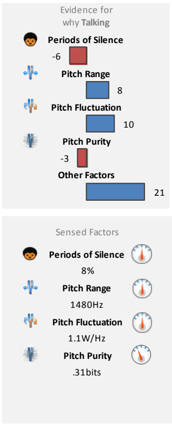
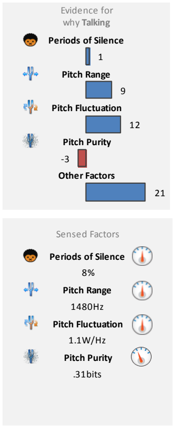
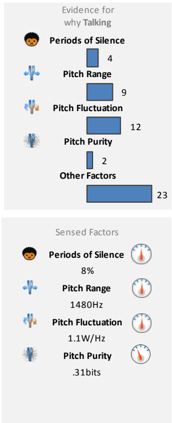
H9 (H3b). After talking to team of coworkers, want to review audio : may not have recorded since inferred as Music.

After a group meeting with some coworkers, you want to review and save the recorded audio. HearMe has a feature to automatically detect speech, and record the last 30 minutes heard. It is supposed to only record if it heard speech, and not other types of sound, so that it does not record useless audio.	Here is an audio clip of what was actually heard at that time: **<AUDIO: h9-talking-work.mp3>**

You review HearMe and see:

Correct Application Response

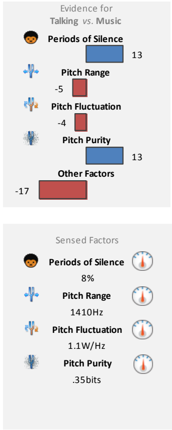
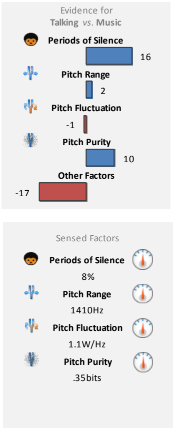
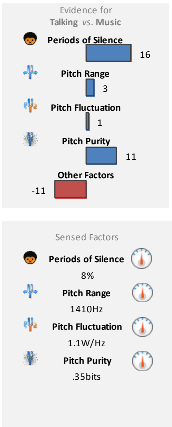
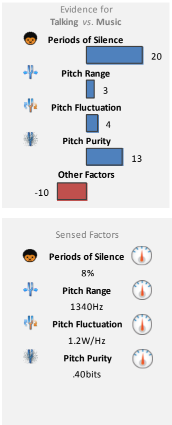
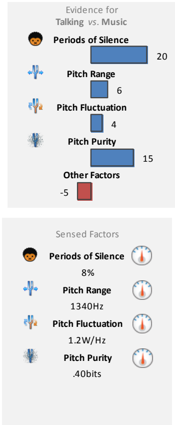
You were **In a Conversation**
Last recorded speech 1 minute ago

50%	60%	70%	80%	90%	100%
					

You then play back the rest of the speech, and save it.

Wrong Application Response

You were **Listening to Music**
Last recorded speech 2 hours ago

50%	60%	70%	80%	90%	100%
					N.A.

You realize that the recent conversation was never recorded.

H10 (H1b). Evening, want to call friend: may be inferred as talking when actually music

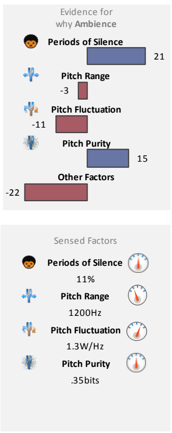
Before you leave work, you want to call your friend, John, to ask to pick up a toolbox you had previously lent him.

You look at HearMe to see his status:

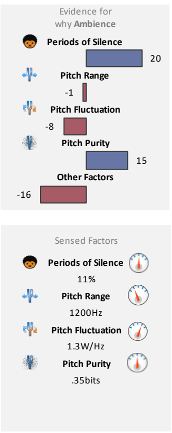
Correct Application Response

John is **Silent**

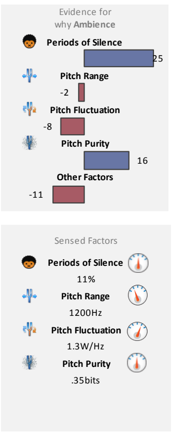
50%



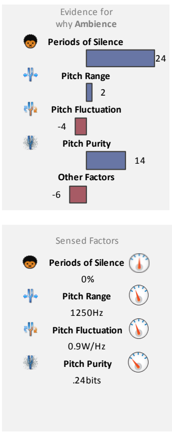
60%



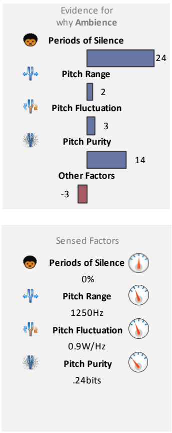
70%



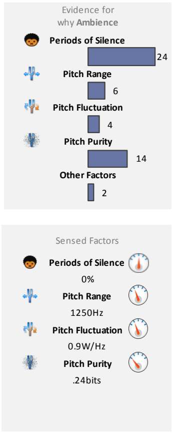
80%



90%



100%

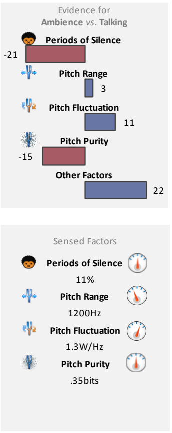


You go ahead to call him, and discuss when you can go over to his place.

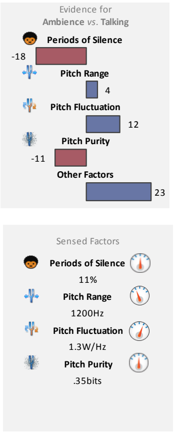
Wrong Application Response

John is **Talking**

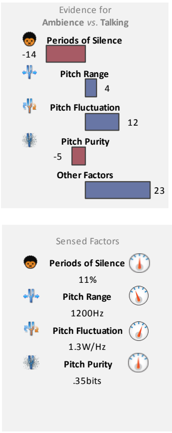
50%



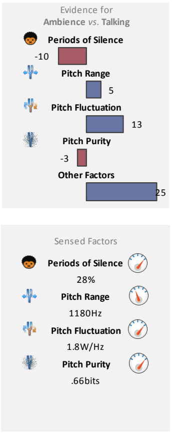
60%



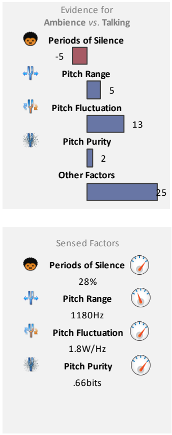
70%



80%



90%



100%

N.A.

You send him a text message to ask him to call you when he is done talking.
But he immediately calls back and tells him he was not talking, and can discuss with you now.

Ground Truth

<NO AUDIO>

F.3 SURVEY INSTRUMENT

We implemented the online survey using LimeSurvey[11]. Participants accessed the survey from Amazon Mechanical Turk through an External Question. LocateMe and HearMe were tested in two separate surveys in two independent tasks.

F.3.1 THINK ALOUD STUDY INSTRUCTIONS

Participants during the think aloud experiment were presented instructions at the beginning of each phase: first version for the non-intelligible condition, second version for the Certainty-only condition, and third version for the Full Intelligibility condition.

[11] LimeSurvey. http://www.limesurvey.org. Retrieved 13 March 2012.

LocateMe - Instructions

First Version

LocateMe detects where you are (*e.g.*, whether at home, in some room in your office, or at some other place), and performs several desirable actions, such as:

- Indicate your availability (*e.g.*, whether you are at the office yet)
- Remind you to get to certain destinations if you are not already there
- Send you notifications or reminders when you are at certain places (*e.g.*, loading the grocery list when it detects you at the supermarket)
- Help you print on the printer nearest to you
- Inform you where your buddies are (*e.g.*, whether at home)

LocateMe uses several sensed factors to estimate where you are. It then infers which place you are (*e.g.* which office room you are in, whether you are at home).
For each scenario, you would be told what is happening in those situations. You may see a map indicating truly where you are (indicated by a star), and indicating where certain named places are. These are provided to help you understand what is happening. **Note that this is not necessarily how LocateMe perceives about the places and where you are. LocateMe may or may not infer your location correctly.**

Second Version

LocateMe detects where you are (*e.g.*, whether at home, in some room in your office, or at some other place), and performs several desirable actions, such as:

- Indicate your availability (*e.g.*, whether you are at the office yet)
- Remind you to get to certain destinations if you are not already there
- Send you notifications or reminders when you are at certain places (*e.g.*, loading the grocery list when it detects you at the supermarket)
- Help you print on the printer nearest to you
- Inform you where your buddies are (*e.g.*, whether at home)

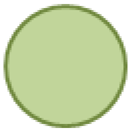
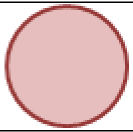
LocateMe uses several sensed factors to estimate where you are. It then infers which place you are (*e.g.* which office room you are in, whether you are at home).

The version of LocateMe you are using can also indicate how confident it is about its inference (*e.g.*, 62%, 93%).

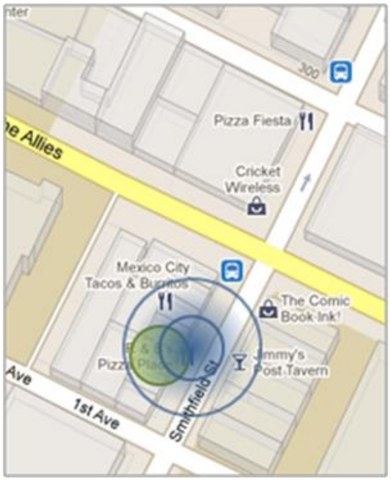
Third Version

Furthermore, it can show you a map with visualizations to show you where it thinks you are and explain to you how it made its inference. It uses several "bubbles" to

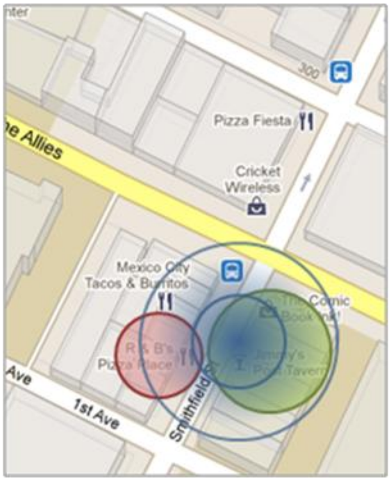
1. Indicate an area where it thinks you could be
2. Indicate the size and location of a named place (*e.g.*, circle representing Home)

Bubble	Description / Meaning
	Indicates the area where LocateMe thinks you could be. You are more likely to be in the center of the circle, but your exact location can be anywhere within the outer circle, and even possibly (though very unlikely) outside it. I.e. the further away from the center of the circle, the less it would think you would be there. The larger the circle, the larger the area that you could be in.
	Represents the area for a named place (*e.g.*, Home, Restaurant, Bar). The size of the circle indicates the threshold of area that would be considered to be. Some places are defined to be larger than others, and some places may be defined to be large to have a lenient threshold. The green color indicates that you inferred to be at this place.
	Also represents an area defined for a named place. However, the red color indicates that you are inferred **not** to be at this place.

Some examples and their interpretations:



This shows that LocateMe infers you to be at R & B's Pizza Place, and actually your location is inferred to be most likely just outside (center of circle).



This shows that LocateMe infers you to be at more likely at Jimmy's Port Tavern, and not R & B's Pizza Place.

HearMe - Instructions

First Version

HearMe detects sounds around you (whether you are in a conversation, listening to music, or just ambient noise), and performs several desirable actions, such as:

- Indicate your availability (*e.g.*, whether you are busy talking to someone)
- Record the detected audio only when it thinks there is talking
- Recommend that you play some music if it detects silence for a long time as you work
- Inform you whether your buddy is busy talking to someone

HearMe uses several sensed factors to evaluate what it hears. It then infers what you could be doing either as talking (being in a conversation), listening to music, or silent (it just hears ambient noise).

Second Version

HearMe detects sounds around you (whether you are in a conversation, listening to music, or just ambient noise), and performs several desirable actions, such as:

- Indicate your availability (*e.g.*, whether you are busy talking to someone)
- Record the detected audio only when it thinks there is talking
- Recommend that you play some music if it detects silence for a long time as you work
- Inform you whether your buddy is busy talking to someone

HearMe uses several sensed factors to evaluate what it hears. It then infers what you could be doing either as talking (being in a conversation), listening to music, or silent (it just hears ambient noise).

The version of HearMe you are using is also able to indicate how confident it is about its inference (e.g., 62%, 93%).

Third Version

Furthermore, it can show you two types of visualizations to explain to you how it made its inference:

1. The **values** of input factors, and for each factor, whether it is below, at, or above the average value for that factor
2. The **evidence** of input factors, whether each factor votes for or against the current inference

The following table describes each factor, and what it means:

Factor	Description / Meaning
Periods of Silence	Indicates the percentage of the sound sampled was relatively silent/quiet than the rest of the sample. [Units: percentage (%)] *The sound of talking would typically have a high percentage of silence.*
Pitch Range	Indicates the range of pitches (frequencies) that was heard in the sound sample, from a low pitch to a high pitch. [Units: Hertz (Hz)] *The sound of Music would typically have a high pitch range.*
Pitch Fluctuation	Indicates, on average, how fast the pitch was fluctuating (changing from high to low, vice-versa). [Units: Watts per Hertz (W/Hz)] *The sound of talking would typically have a high rate of pitch fluctuation, while the sound of Music would typically have a low rate.*
Pitch Purity	Indicates how "pure" or "noisy"/"impure" the sound heard was. [Units: bits] *The sound of talking would typically have high pitch purity.*
Other Factors	There are many other more technical factors that are used for sound inference, and they are summarized as "Other Factors." Their exact values are not shown, but they contribute evidence to the inference.

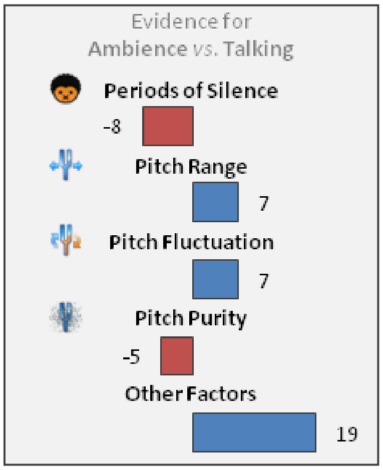
HearMe can tell you whether the sensed values of these factors are below, at, or above the average value for that factor, across all sound samples it has heard. This is indicated with gauge icons:

Much Below Average	Below Average	Average	Above Average	Much Above Average
⏲	⏲	⏲	⏲	⏲

Other than telling you the values of the factors, HearMe can also tell you how much evidence each factor votes for or against an inference. This is a numerical point, either positive, negative, or zero. The average of all these weights of evidence leads to the ultimate inference. These weights are shown in a bar chart with numbers at the ends of the bars.

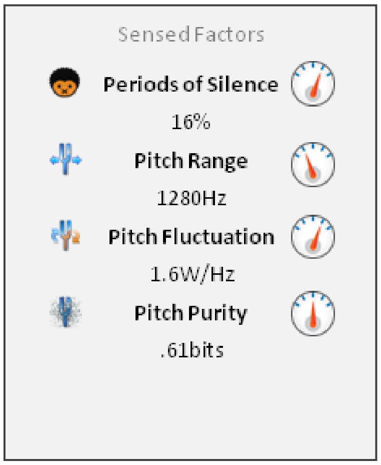
Some examples and their interpretations:

Evidence Visualization



On average, the weights of factors vote more for inferring Talking, instead of Ambience. The strongest factor voting this direction is **Other Factors** with 19 points. However, **Periods of Silence** and **Pitch Purity** vote for Ambience with -8 and -5 points, respectively.

Sensed Factors Visualization



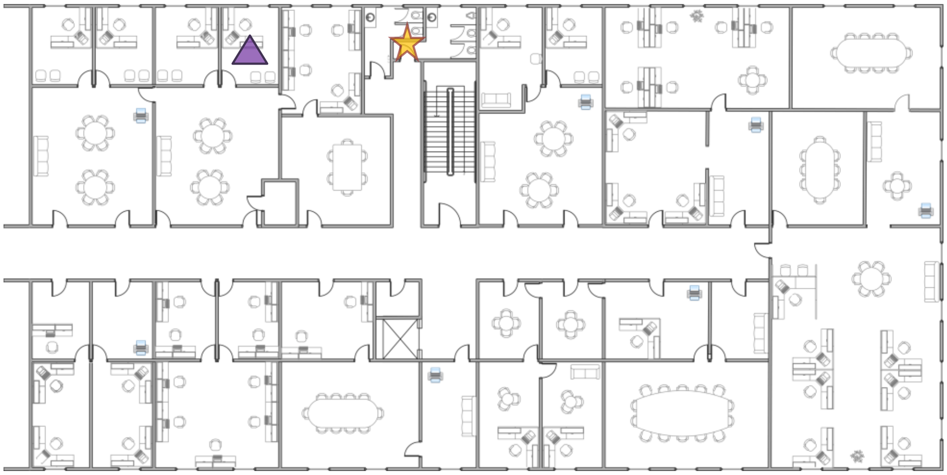
Periods of Silence: 16% of the sound sampled was relatively silent compared to the rest of the sample. This is above average for that factor. **Pitch Purity**: the sound sampled has a purity of .61bits, which is about the average for samples heard.

F.3.2 LOCATEME SCENARIOS ACROSS INTELLIGIBILITY CONDITIONS

LocateMe (1st)

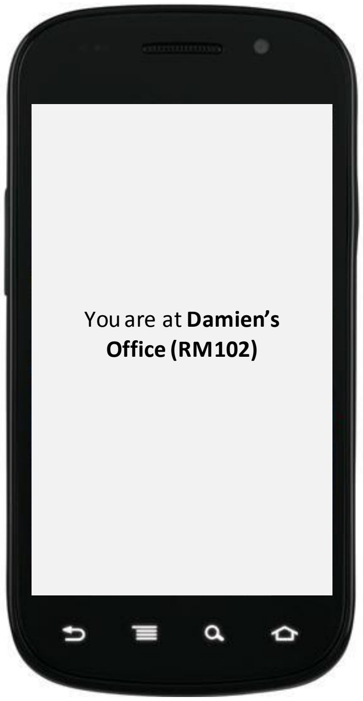
Scenario A

You are in the washroom just before your meeting with your neighboring coworker, Damien.



The star denotes where you actually are for this scenario.
The purple triangle denotes where Damien's office (RM102) is.

You receive a call from your neighboring coworker, Damien, asking you where you are since LocateMe told him that you are at his office but he obviously does not see you there.
You check your status and see:

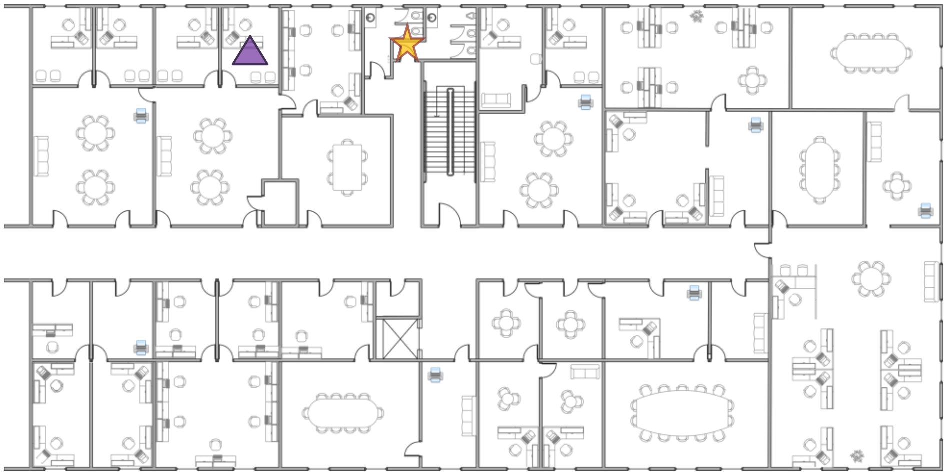


You are at **Damien's Office (RM102)**

LocateMe (2nd)

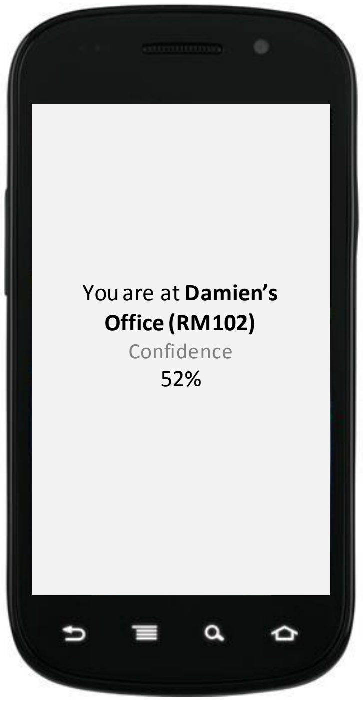
Scenario A

You are in the washroom just before your meeting with your neighboring coworker, Damien.



The star denotes where you actually are for this scenario.
The purple triangle denotes where Damien's office (RM102) is.

You receive a call from your neighboring coworker, Damien, asking you where you are since LocateMe told him that you are at his office but he obviously does not see you there.
You check your status and see:

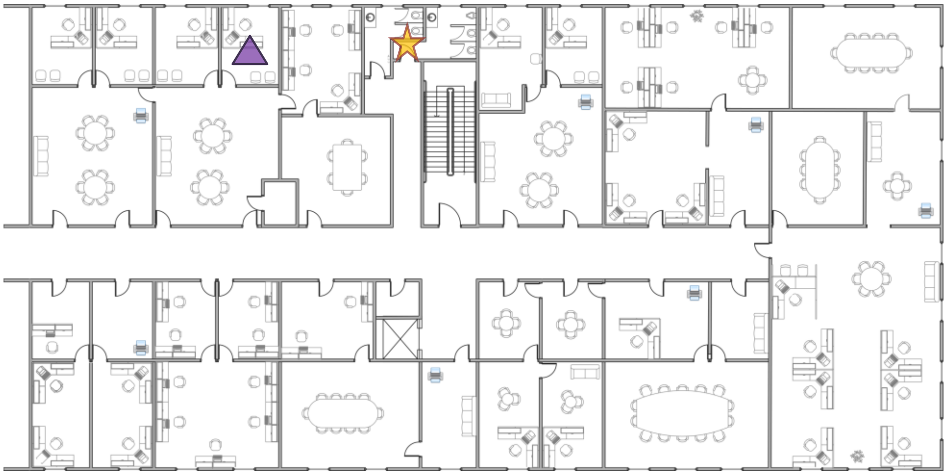


You are at **Damien's Office (RM102)**
Confidence
52%

LocateMe (3rd)

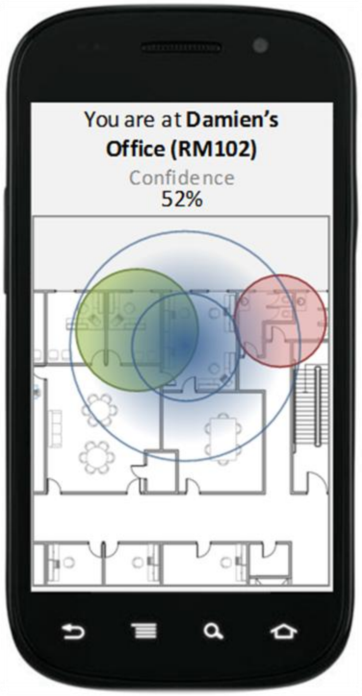
Scenario A

You are in the washroom just before your meeting with your neighboring coworker, Damien.



The star denotes where you actually are for this scenario.
The purple triangle denotes where Damien's office (RM102) is.

You receive a call from your neighboring coworker, Damien, asking you where you are since LocateMe told him that you are at his office but he obviously does not see you there.
You check your status and see:



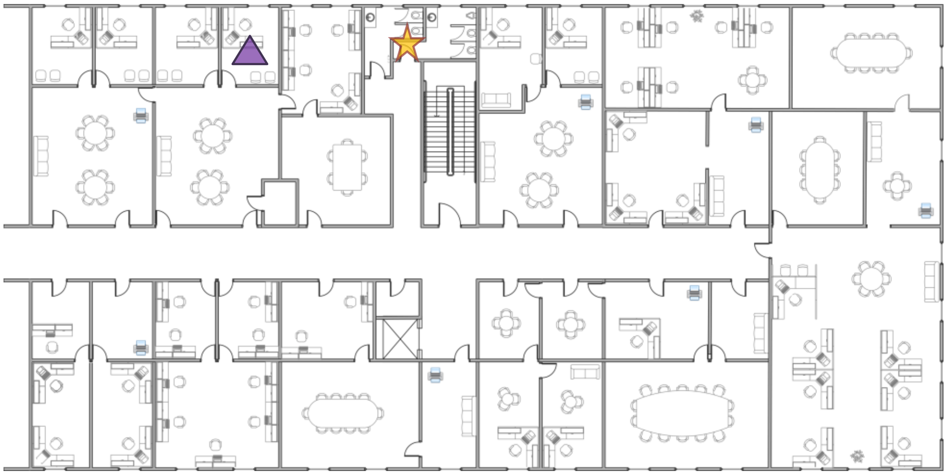
You are at **Damien's Office (RM102)**

Confidence
52%

LocateMe (1st) ^

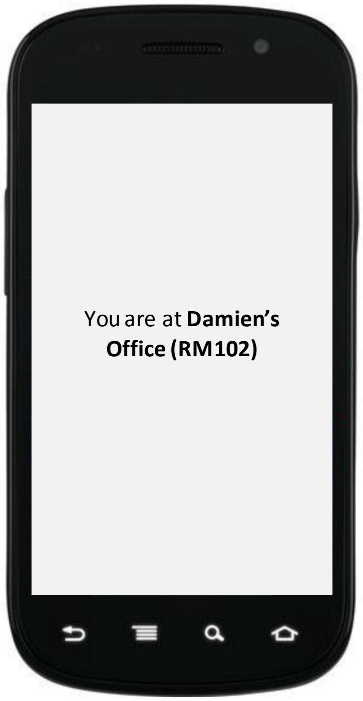
Scenario A

You are in the washroom just before your meeting with your neighboring coworker, Damien.



The star denotes where you actually are for this scenario.
The purple triangle denotes where Damien's office (RM102) is.

You receive a call from your neighboring coworker, Damien, asking you where you are since LocateMe told him that you are at his office but he obviously does not see you there.
You check your status and see:

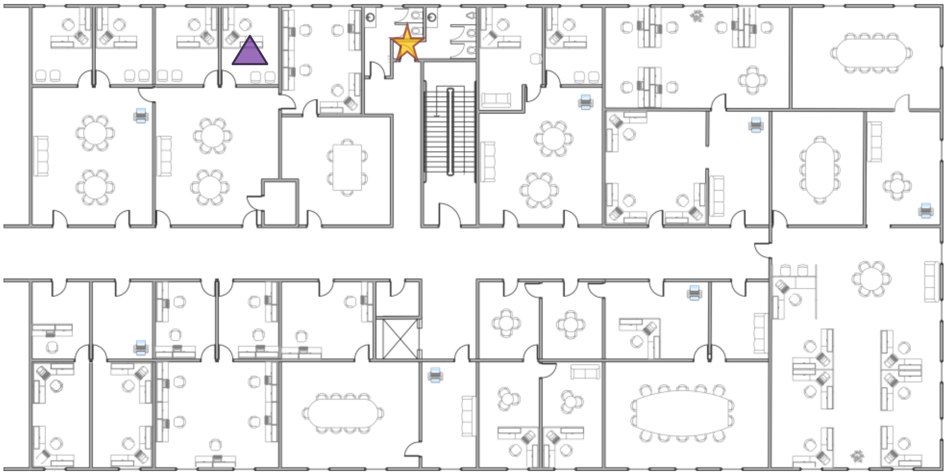


You are at **Damien's Office (RM102)**

LocateMe (2nd) ^

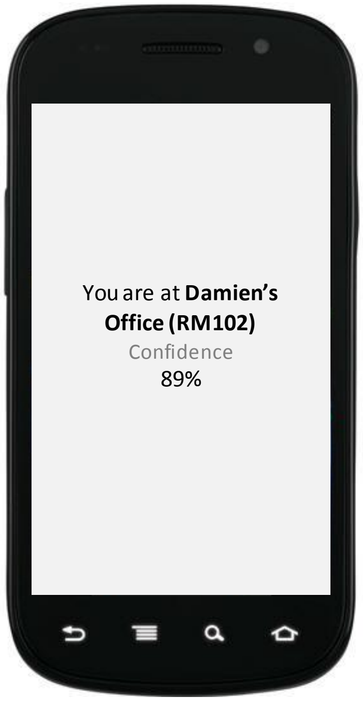
Scenario A

You are in the washroom just before your meeting with your neighboring coworker, Damien.



The star denotes where you actually are for this scenario.
The purple triangle denotes where Damien's office (RM102) is.

You receive a call from your neighboring coworker, Damien, asking you where you are since LocateMe told him that you are at his office but he obviously does not see you there.
You check your status and see:

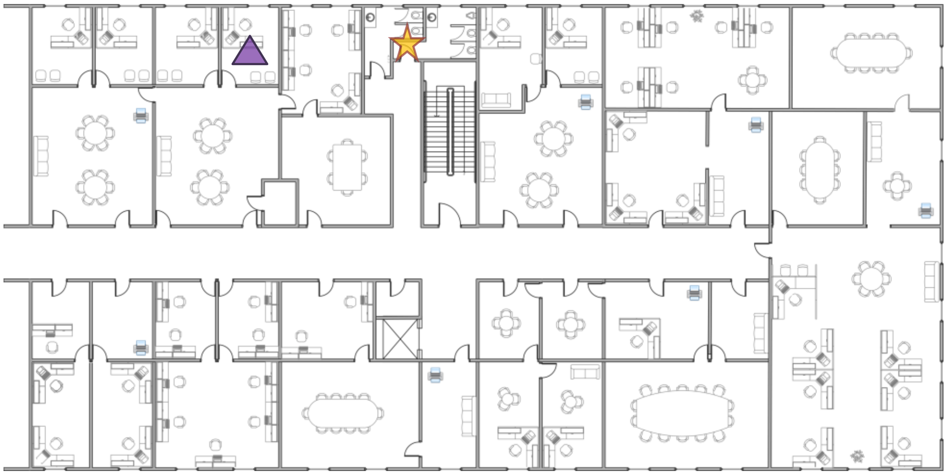


You are at **Damien's Office (RM102)**
Confidence
89%

LocateMe (3rd) ^

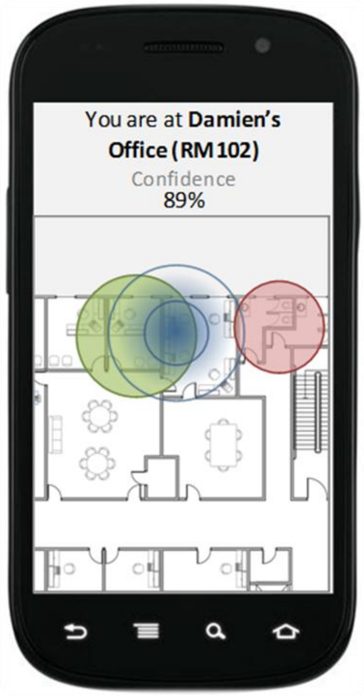
Scenario A

You are in the washroom just before your meeting with your neighboring coworker, Damien.



The star denotes where you actually are for this scenario.
The purple triangle denotes where Damien's office (RM102) is.

You receive a call from your neighboring coworker, Damien, asking you where you are since LocateMe told him that you are at his office but he obviously does not see you there.
You check your status and see:

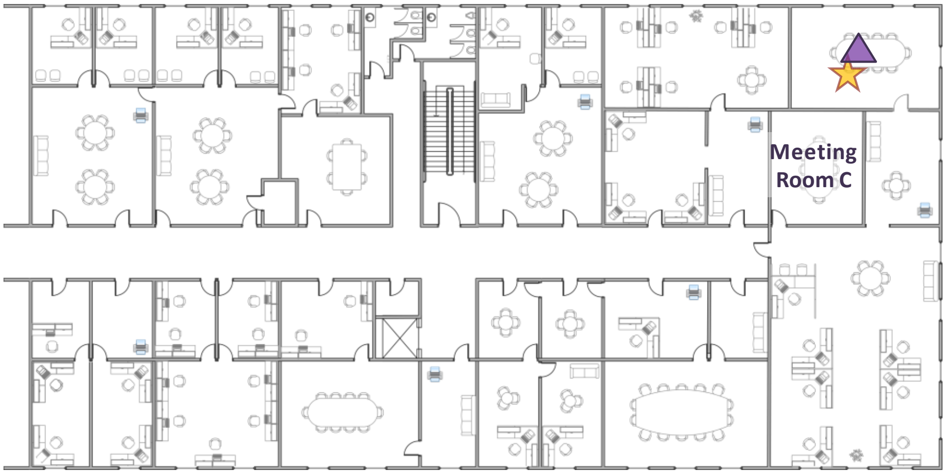


You are at **Damien's Office (RM 102)**
Confidence
89%

LocateMe (1st)

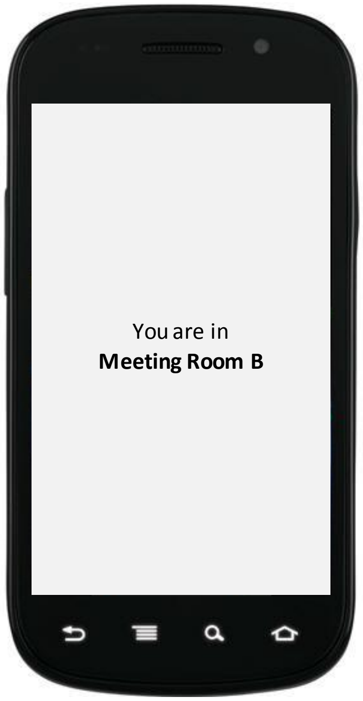
Scenario B

You arrive in Meeting Room B five minutes early.



The star denotes where you actually are for this scenario.
The purple triangle denotes where Meeting Room B is.

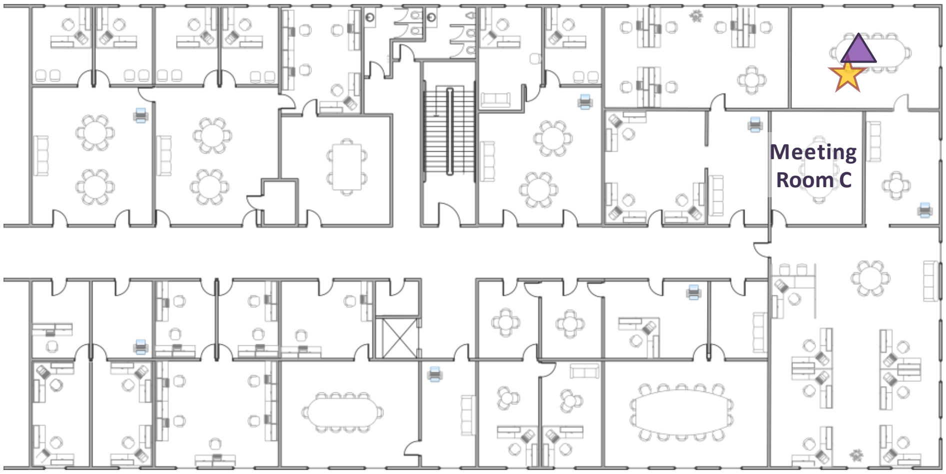
LocateMe beeps and loads a document containing the meeting agenda.
You check your status and see:



You are in
Meeting Room B

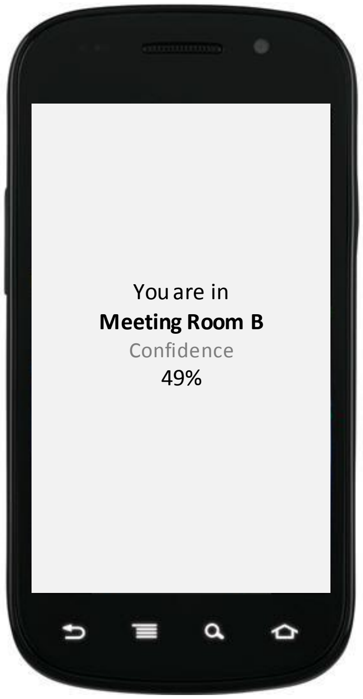
LocateMe (2nd)
Scenario B

You arrive in Meeting Room B five minutes early.



The star denotes where you actually are for this scenario.
The purple triangle denotes where Meeting Room B is.

LocateMe beeps and loads a document containing the meeting agenda.
You check your status and see:

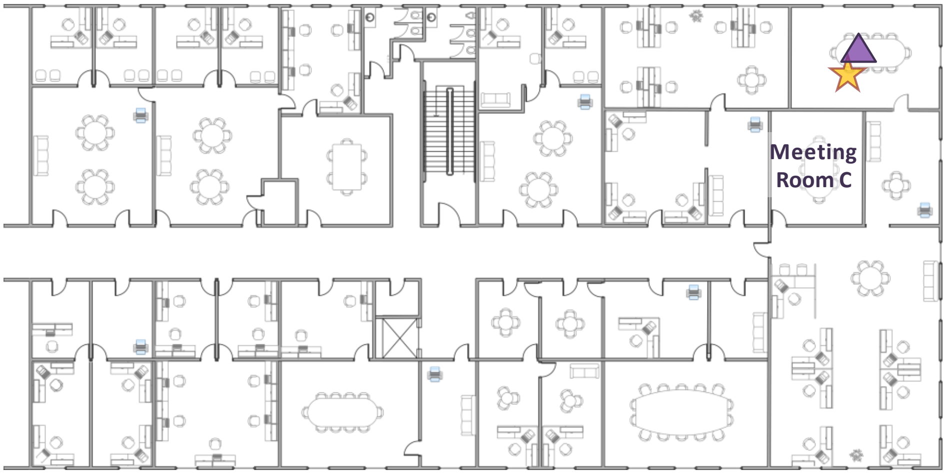


You are in
Meeting Room B
Confidence
49%

LocateMe (3rd)

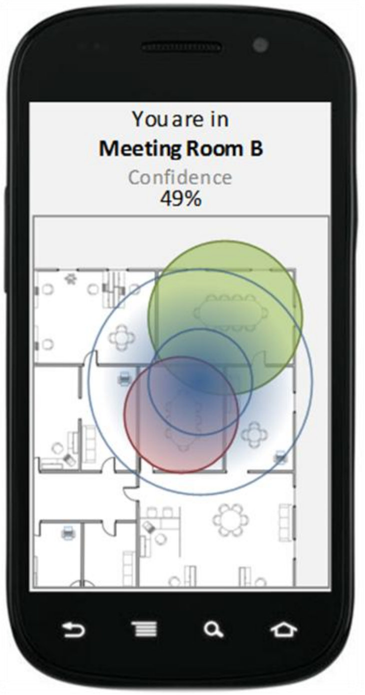
Scenario B

You arrive in Meeting Room B five minutes early.



The star denotes where you actually are for this scenario.
The purple triangle denotes where Meeting Room B is.

LocateMe beeps and loads a document containing the meeting agenda.
You check your status and see:

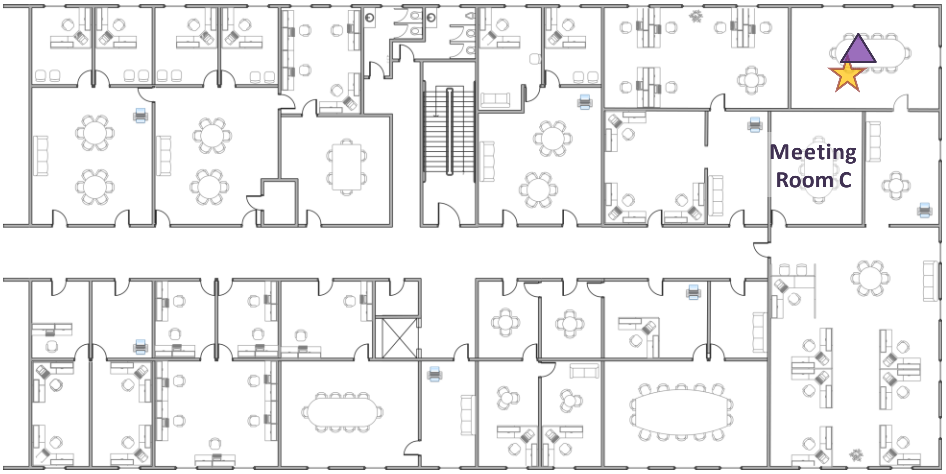


You are in
Meeting Room B
Confidence
49%

LocateMe (1st) ^

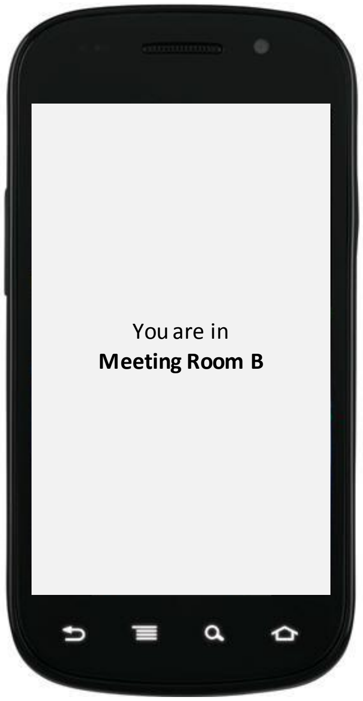
Scenario B

You arrive in Meeting Room B five minutes early.



The star denotes where you actually are for this scenario.
The purple triangle denotes where Meeting Room B is.

LocateMe beeps and loads a document containing the meeting agenda.
You check your status and see:

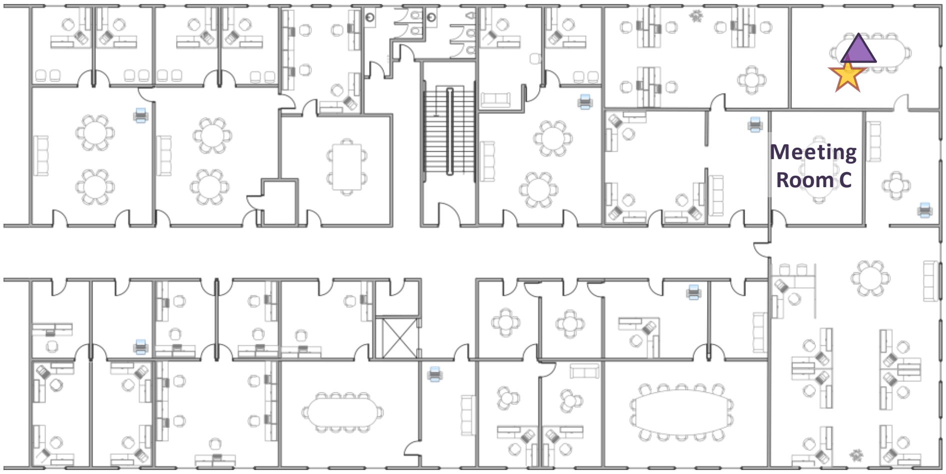


You are in
Meeting Room B

LocateMe (2nd) ^

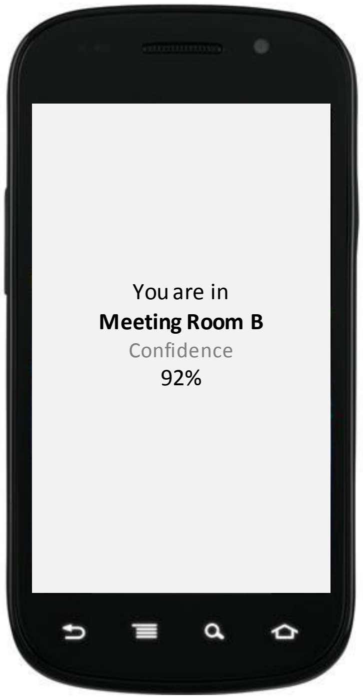
Scenario B

You arrive in Meeting Room B five minutes early.



The star denotes where you actually are for this scenario.
The purple triangle denotes where Meeting Room B is.

LocateMe beeps and loads a document containing the meeting agenda.
You check your status and see:

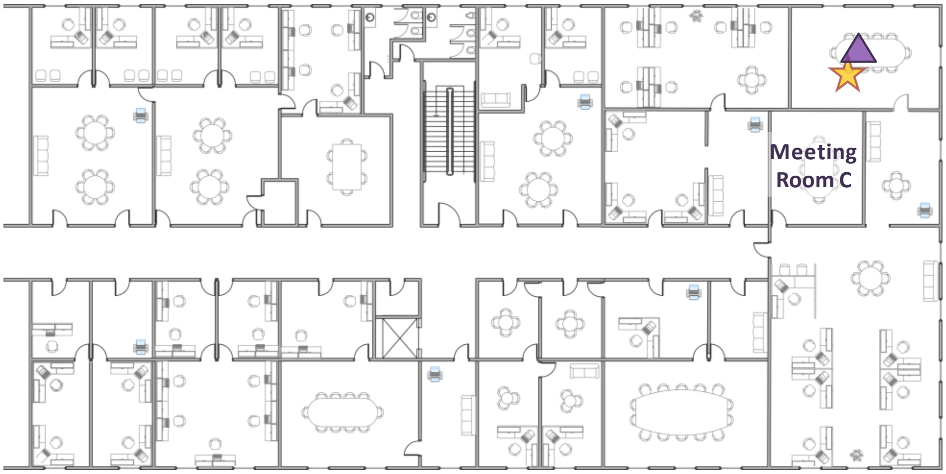


You are in
Meeting Room B
Confidence
92%

LocateMe (3rd) ^

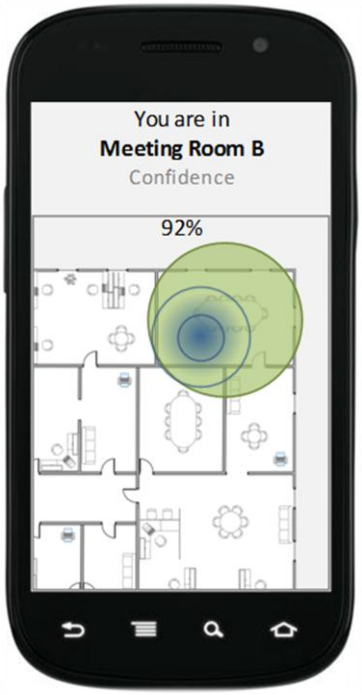
Scenario B

You arrive in Meeting Room B five minutes early.



The star denotes where you actually are for this scenario.
The purple triangle denotes where Meeting Room B is.

LocateMe beeps and loads a document containing the meeting agenda.
You check your status and see:



You are in
Meeting Room B
Confidence
92%

F.3.3 HearMe Scenarios across Intelligibility Conditions

HearMe (1st)

Scenario A

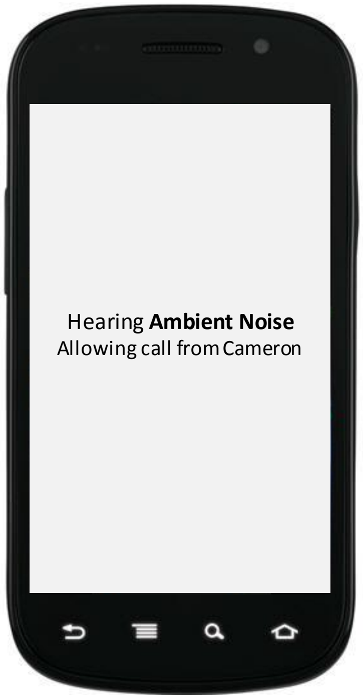
At the coffee shop, you find your coworker Michelle there, and have a chat with her.

Play audio clip of what was actually heard at that time.

▶

After 5 minutes, midway through your conversation, you get an interruption from another coworker, Cameron, calling you on your phone.

You quickly ignore the call, and check HearMe's status of you:



Hearing **Ambient Noise**
Allowing call from Cameron

HearMe (2nd)

Scenario A

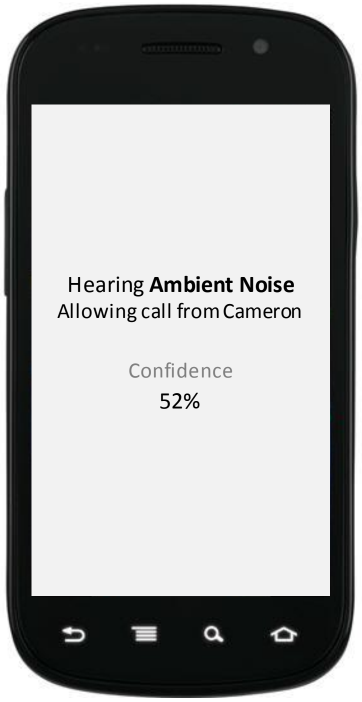
At the coffee shop, you find your coworker Michelle there, and have a chat with her.

Play audio clip of what was actually heard at that time.

▶

After 5 minutes, midway through your conversation, you get an interruption from another coworker, Cameron, calling you on your phone.

You quickly ignore the call, and check HearMe's status of you:



Hearing **Ambient Noise**
Allowing call from Cameron

Confidence
52%

HearMe (3rd)

Scenario A

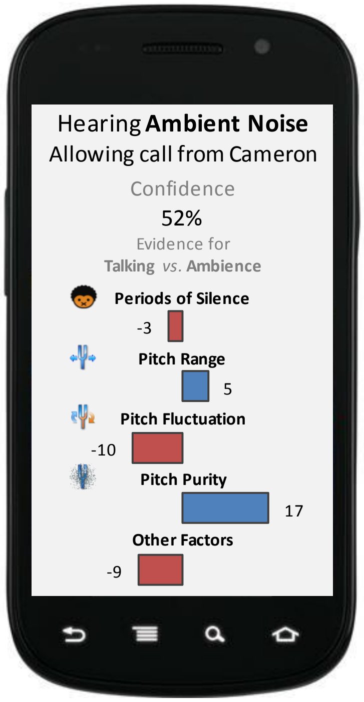
At the coffee shop, you find your coworker Michelle there, and have a chat with her.

Play audio clip of what was actually heard at that time.

▶

After 5 minutes, midway through your conversation, you get an interruption from another coworker, Cameron, calling you on your phone.

You quickly ignore the call, and check HearMe's status of you:



Hearing **Ambient Noise**
Allowing call from Cameron

Confidence
52%

Evidence for
Talking *vs.* **Ambience**

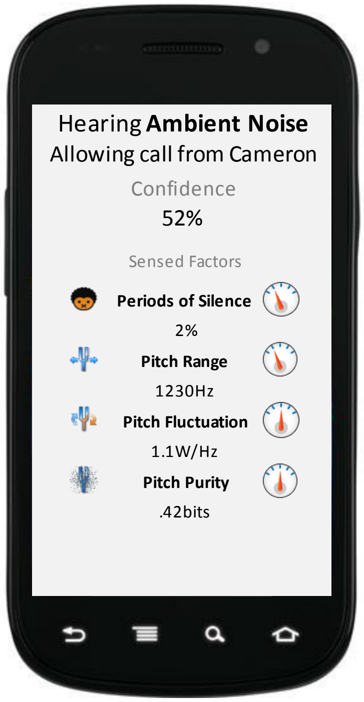
Periods of Silence
-3

Pitch Range
5

Pitch Fluctuation
-10

Pitch Purity
17

Other Factors
-9



Hearing **Ambient Noise**
Allowing call from Cameron

Confidence
52%

Sensed Factors

Periods of Silence
2%

Pitch Range
1230Hz

Pitch Fluctuation
1.1W/Hz

Pitch Purity
.42bits

HearMe (1st) ^

Scenario A

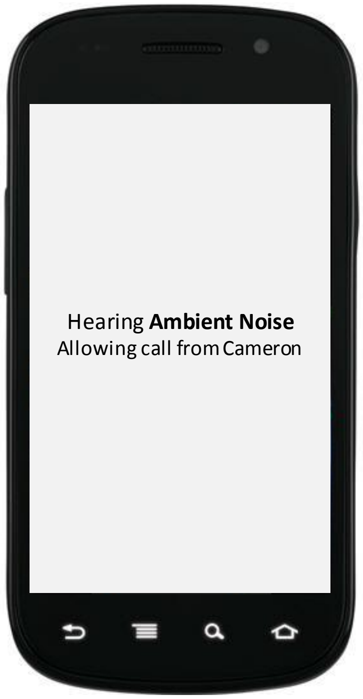
At the coffee shop, you find your coworker Michelle there, and have a chat with her.

Play audio clip of what was actually heard at that time.

▶

After 5 minutes, midway through your conversation, you get an interruption from another coworker, Cameron, calling you on your phone.

You quickly ignore the call, and check HearMe's status of you:



Hearing **Ambient Noise**
Allowing call from Cameron

HearMe (2nd) ^

Scenario A

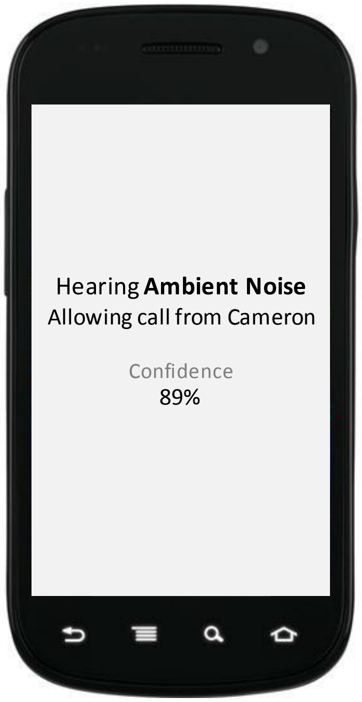
At the coffee shop, you find your coworker Michelle there, and have a chat with her.

Play audio clip of what was actually heard at that time.

▶

After 5 minutes, midway through your conversation, you get an interruption from another coworker, Cameron, calling you on your phone.

You quickly ignore the call, and check HearMe's status of you:



Hearing **Ambient Noise**
Allowing call from Cameron

Confidence
89%

HearMe (3rd) ^

Scenario A

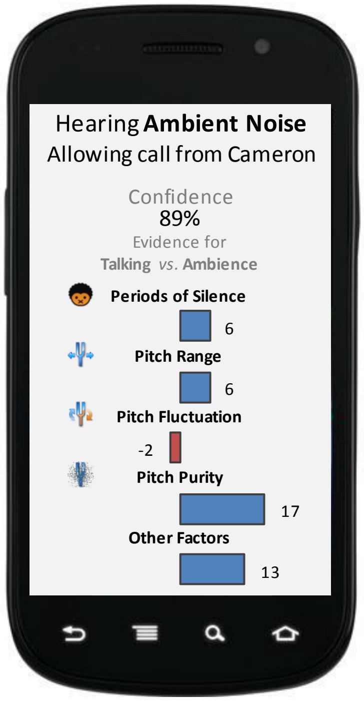
At the coffee shop, you find your coworker Michelle there, and have a chat with her.

Play audio clip of what was actually heard at that time.

▶

After 5 minutes, midway through your conversation, you get an interruption from another coworker, Cameron, calling you on your phone.

You quickly ignore the call, and check HearMe's status of you:



Hearing **Ambient Noise**
Allowing call from Cameron

Confidence
89%

Evidence for
Talking *vs.* **Ambience**

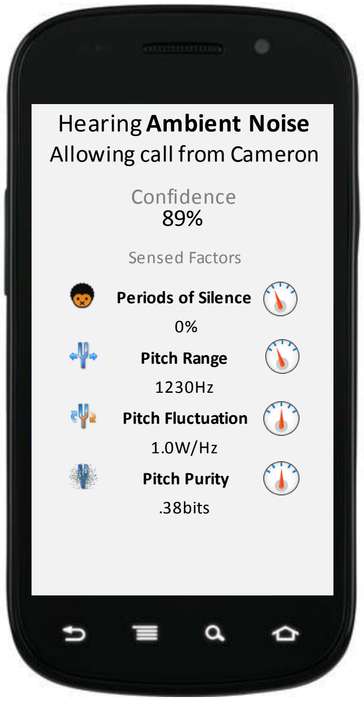
Periods of Silence 6

Pitch Range 6

Pitch Fluctuation -2

Pitch Purity 17

Other Factors 13



Hearing **Ambient Noise**
Allowing call from Cameron

Confidence
89%

Sensed Factors

Periods of Silence
0%

Pitch Range
1230Hz

Pitch Fluctuation
1.0W/Hz

Pitch Purity
.38bits

HearMe (1st)

Scenario B

After a group meeting with some coworkers, you want to review and save the recorded audio.

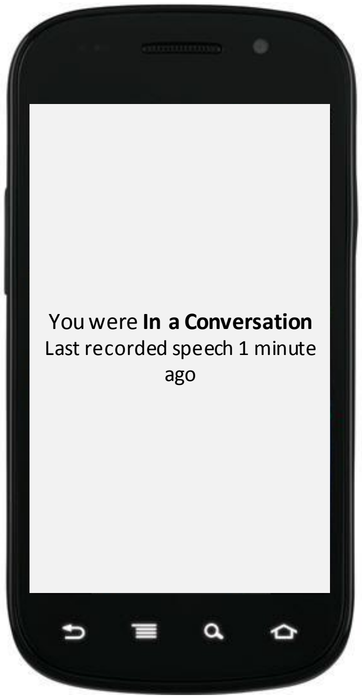
Play audio clip of what was actually heard at that time.

▶

HearMe has a feature to automatically detect speech, and record the last 30 minutes heard.

It is supposed to only record if it heard speech, and not other types of sound, so that it does not record useless audio.

You review HearMe and see:.



You were **In a Conversation**
Last recorded speech 1 minute ago

You then play back the rest of the speech, and save it.

HearMe (2nd)

Scenario B

After a group meeting with some coworkers, you want to review and save the recorded audio.

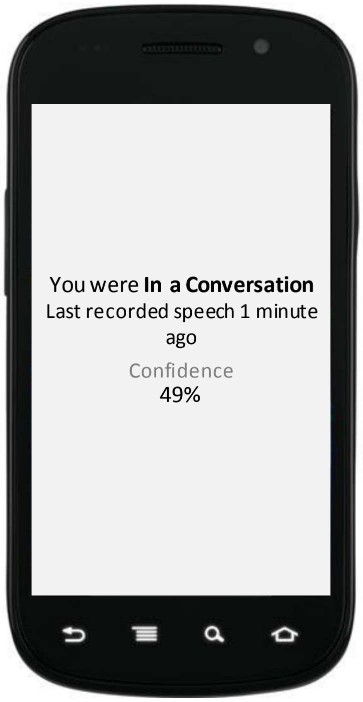
Play audio clip of what was actually heard at that time.

▶

HearMe has a feature to automatically detect speech, and record the last 30 minutes heard.

It is supposed to only record if it heard speech, and not other types of sound, so that it does not record useless audio.

You review HearMe and see:.



You were **In a Conversation**
Last recorded speech 1 minute ago
Confidence
49%

You then play back the rest of the speech, and save it.

HearMe (3rd)

Scenario B

After a group meeting with some coworkers, you want to review and save the recorded audio.

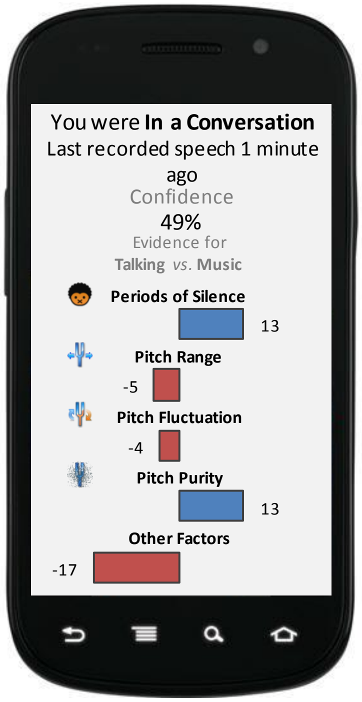
Play audio clip of what was actually heard at that time.

▶

HearMe has a feature to automatically detect speech, and record the last 30 minutes heard.

It is supposed to only record if it heard speech, and not other types of sound, so that it does not record useless audio.

You review HearMe and see:.



You were **In a Conversation**
Last recorded speech 1 minute ago
Confidence
49%
Evidence for
Talking *vs.* **Music**

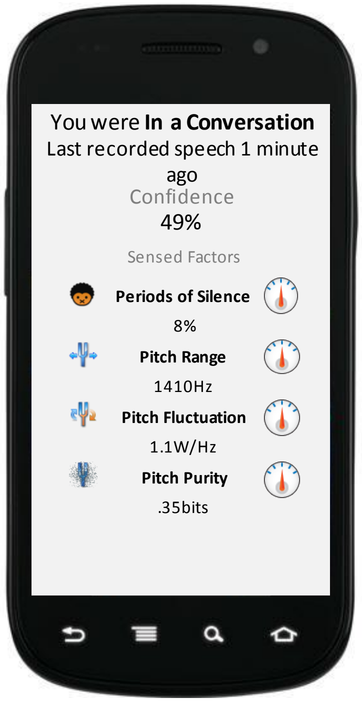
Periods of Silence 13

Pitch Range -5

Pitch Fluctuation -4

Pitch Purity 13

Other Factors -17



You were **In a Conversation**
Last recorded speech 1 minute ago
Confidence
49%

Sensed Factors

Periods of Silence 8%

Pitch Range 1410Hz

Pitch Fluctuation 1.1W/Hz

Pitch Purity .35bits

You then play back the rest of the speech, and save it.

HearMe (1st) ^

Scenario B

After a group meeting with some coworkers, you want to review and save the recorded audio.

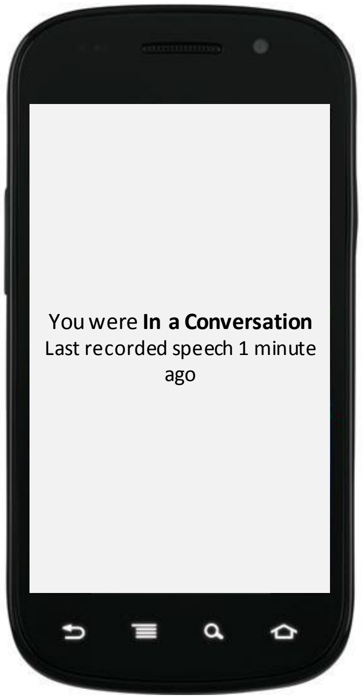
Play audio clip of what was actually heard at that time.

▶

HearMe has a feature to automatically detect speech, and record the last 30 minutes heard.

It is supposed to only record if it heard speech, and not other types of sound, so that it does not record useless audio.

You review HearMe and see:.



You were **In a Conversation**
Last recorded speech 1 minute ago

You then play back the rest of the speech, and save it.

HearMe (2nd) ^

Scenario B

After a group meeting with some coworkers, you want to review and save the recorded audio.

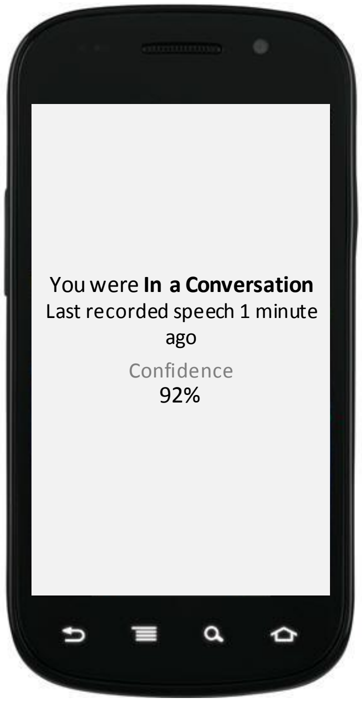
Play audio clip of what was actually heard at that time.

▶

HearMe has a feature to automatically detect speech, and record the last 30 minutes heard.

It is supposed to only record if it heard speech, and not other types of sound, so that it does not record useless audio.

You review HearMe and see:.



You were **In a Conversation**
Last recorded speech 1 minute ago
Confidence
92%

You then play back the rest of the speech, and save it.

HearMe (3rd) ^

Scenario B

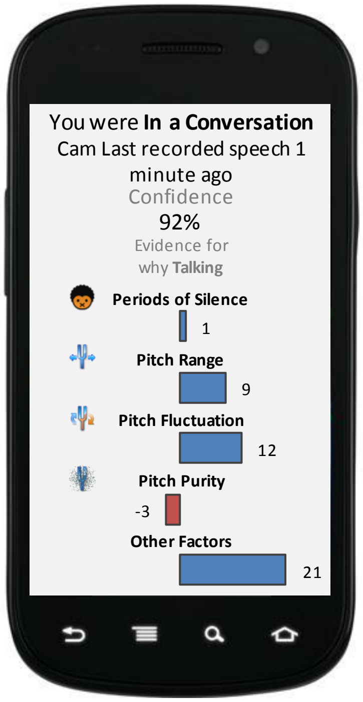
After a group meeting with some coworkers, you want to review and save the recorded audio.

Play audio clip of what was actually heard at that time.

HearMe has a feature to automatically detect speech, and record the last 30 minutes heard.

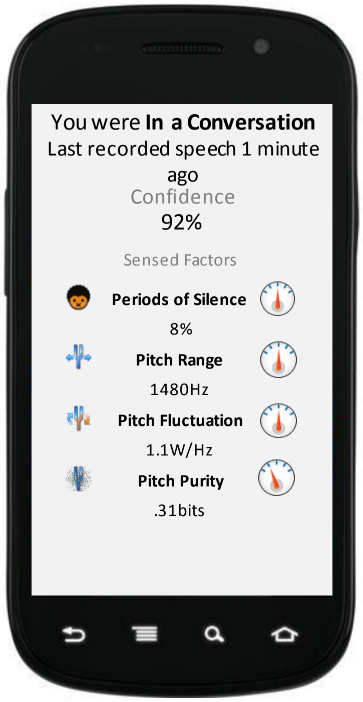
It is supposed to only record if it heard speech, and not other types of sound, so that it does not record useless audio.

You review HearMe and see:.



You were **In a Conversation**
Cam Last recorded speech 1 minute ago
Confidence
92%
Evidence for why **Talking**

Periods of Silence
1
Pitch Range
9
Pitch Fluctuation
12
Pitch Purity
-3
Other Factors
21



You were **In a Conversation**
Last recorded speech 1 minute ago
Confidence
92%

Sensed Factors

Periods of Silence
8%
Pitch Range
1480Hz
Pitch Fluctuation
1.1W/Hz
Pitch Purity
.31bits

You then play back the rest of the speech, and save it.

F.3.3.1 SURVEY QUESTIONS FOR EACH SCENARIO

Participant #: __________ Time and Date: ___________________________
Gender: _________, Age: __________, Education: __________________, Occupation: __________________
Application: ___________________, Scenario: _______

1. What is happening in this situation?
2. How **important** or **unimportant** do you feel that the application should behave appropriately in this situation?

	1st	2nd	3rd
Very Unimportant			
Unimportant			
Somewhat Unimportant			
Neither Unimportant nor Important			
Somewhat Important			
Important			
Very Important			

3. What did the application do in this situation?
4. How **confident** do you think the application is of its inference?

	1st	2nd	3rd
Give a percentage between 0 and 100%			

5. What did the application do in this situation?
6. How **appropriately** or **inappropriately** did the application behave in this situation?

	1st	2nd	3rd
Very Inappropriately			
Inappropriately			
Somewhat Inappropriately			
Neither Inappropriately nor Appropriately			
Somewhat Appropriately			
Appropriately			
Very Appropriately			

7. How much do you **agree** or **disagree** with the application's inference, given how easy or difficult it is to infer this?

	1st	**2nd**	**3rd**
Strongly Disagree			
Disagree			
Somewhat Disagree			
Neither Disagree nor Agree			
Somewhat Agree			
Agree			
Strongly Agree			

8. **Why** did the application infer what it did? (*Experimenter elaborates*)

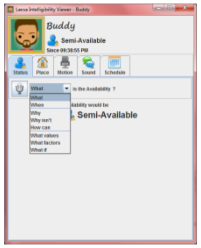
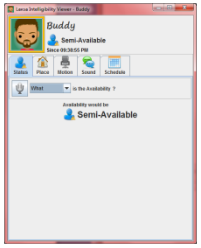
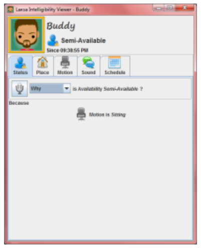
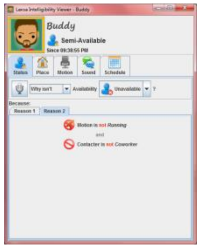
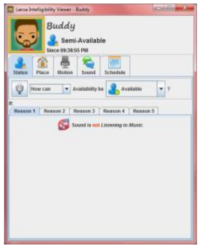
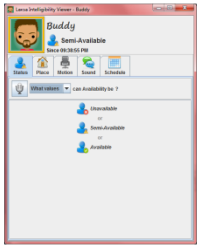
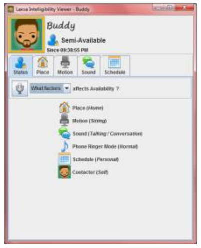
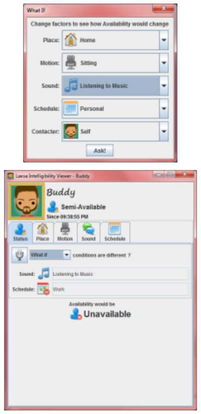
G LAKSA USER INTERFACE

This appendix describes the user interface of various aspects of the Laκsa prototype used in the design and usability study in Chapter 7.

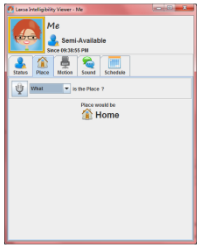
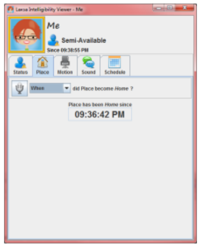
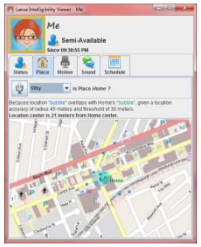
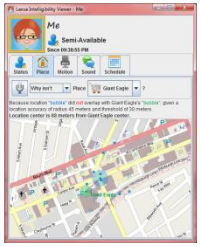
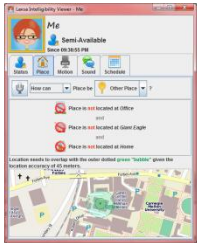
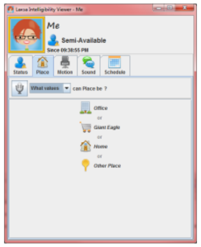
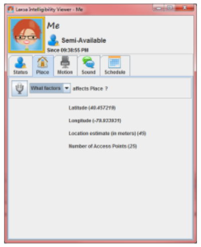
G.1 Intelligibility User Interface

Example screenshots of intelligibility explanation types provided by Laksa. The GUI was implemented for the desktop in Java Swing.

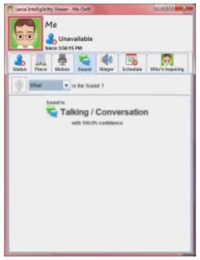
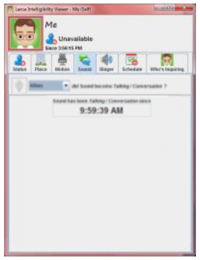
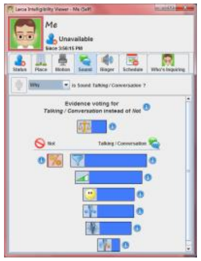
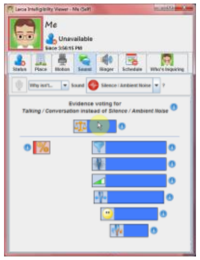
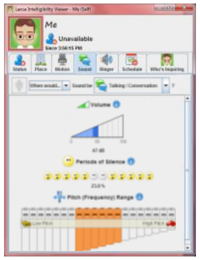
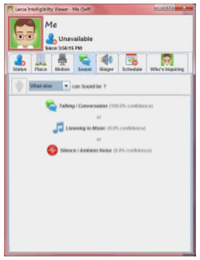
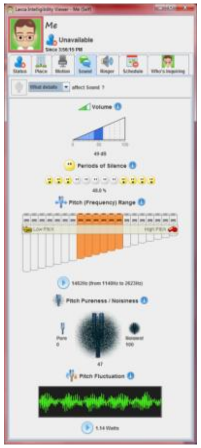
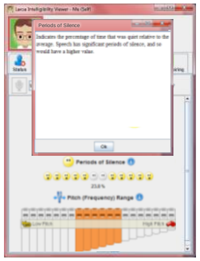
G.1.1 Availability

What	When	Why	Why Not	How Can	Outputs	Inputs	What If
 Laksa Intelligibility Viewer - Buddy Buddy Semi-Available Since 09:38:55 PM Status Place Motion Sound Schedule What When Why Why isn't How can What values What factors What if What is the Availability ? Availability would be Semi-Available	 Laksa Intelligibility Viewer - Buddy Buddy Semi-Available Since 09:38:55 PM Status Place Motion Sound Schedule What is the Availability ? Availability would be Semi-Available	 Laksa Intelligibility Viewer - Buddy Buddy Semi-Available Since 09:38:55 PM Status Place Motion Sound Schedule Why is Availability Semi-Available ? Because Motion is Sitting	 Laksa Intelligibility Viewer - Buddy Buddy Semi-Available Since 09:38:55 PM Status Place Motion Sound Schedule Reason 1 Reason 2 Why isn't Availability Unavailable ? Because: Motion is not Running and Contacter is not Coworker	 Laksa Intelligibility Viewer - Buddy Buddy Semi-Available Since 09:38:55 PM Status Place Motion Sound Schedule Reason 1 Reason 2 Reason 3 Reason 4 Reason 5 How can Availability be Available ? If: Sound is not Listening to Music	 Laksa Intelligibility Viewer - Buddy Buddy Semi-Available Since 09:38:55 PM Status Place Motion Sound Schedule What values can Availability be ? Unavailable or Semi-Available or Available	 Laksa Intelligibility Viewer - Buddy Buddy Semi-Available Since 09:38:55 PM Status Place Motion Sound Schedule What factors affects Availability ? Place (Home) Motion (Sitting) Sound (Talking / Conversation) Phone Ringer Mode (Normal) Schedule (Personal) Contacter (Self)	 What If Ask! Laksa Intelligibility Viewer - Buddy Buddy Semi-Available Since 09:38:55 PM Status Place Motion Sound Schedule Change factors to see how Availability would change Place: Home Motion: Sitting Sound: Listening to Music Schedule: Personal Contacter: Self What if conditions are different ? Sound: Listening to Music Schedule: Work Availability would be Unavailable

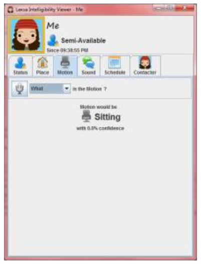
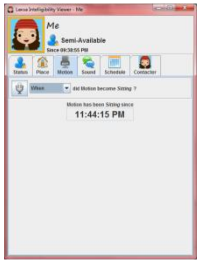
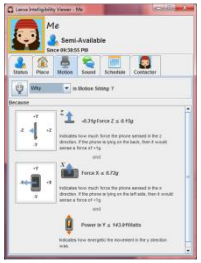
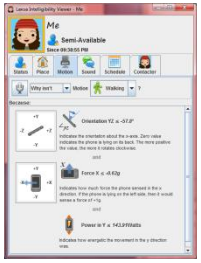
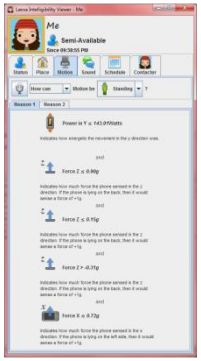
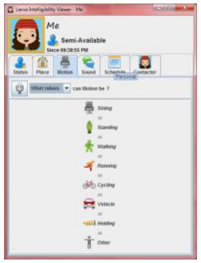
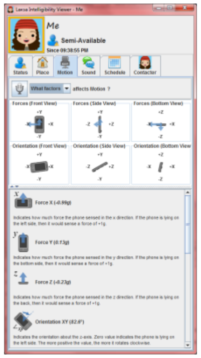
G.1.2 Location

What	When	Why	Why Not	How Can	Outputs	Inputs
 Laksa Intelligibility Viewer - Me Me Semi-Available Since 09:38:55 PM Status Place Motion Sound Schedule What is the Place ? Place would be Home	 Laksa Intelligibility Viewer - Me Me Semi-Available Since 09:38:55 PM Status Place Motion Sound Schedule When did Place become Home ? Place has been Home since 09:36:42 PM	 Laksa Intelligibility Viewer - Me Me Semi-Available Since 09:38:55 PM Status Place Motion Sound Schedule Home Why is Place Home ? Because location "bubble" overlaps with Home's "bubble", given a location accuracy of radius 45 meters and threshold of 30 meters. Location center is 21 meters from Home center.	 Laksa Intelligibility Viewer - Me Me Semi-Available Since 09:38:55 PM Status Place Motion Sound Schedule Giant Eagle Why isn't Place Giant Eagle ? Because location "bubble" did not overlap with Giant Eagle's "bubble", given a location accuracy of radius 45 meters and threshold of 30 meters. Location center is 88 meters from Giant Eagle center.	 Laksa Intelligibility Viewer - Me Me Semi-Available Since 09:38:55 PM Status Place Motion Sound Schedule Forbes Ave Carnegie Mellon University How can Place be Other Place ? Place is not located at Office and Place is not located at Giant Eagle and Place is not located at Home Location needs to overlap with the outer dotted green "bubble" given the location accuracy of 45 meters.	 Laksa Intelligibility Viewer - Me Me Semi-Available Since 09:38:55 PM Status Place Motion Sound Schedule What values can Place be ? Office or Giant Eagle or Home or Other Place	 Laksa Intelligibility Viewer - Me Me Semi-Available Since 09:38:55 PM Status Place Motion Sound Schedule What factors affects Place ? Latitude (40.457219) Longitude (-79.933931) Location estimate (in meters) (45) Number of Access Points (25)

G.1.3 Sound

What	When	Why	Why Not	How Can	Outputs	Inputs	Definition
 Me Unavailable Since 3:56:15 PM What is the Sound ? Sound is Talking / Conversation with 100.0% confidence	 Me Unavailable Since 3:56:15 PM When did Sound become Talking / Conversation ? Sound has been Talking / Conversation since 9:59:39 AM	 Me Unavailable Since 3:56:15 PM Why is Sound Talking / Conversation ? Evidence voting for *Talking / Conversation instead of Not* Not Talking / Conversation	 Me Unavailable Since 3:56:15 PM Why isn't... Sound Silence / Ambient Noise ? Evidence voting for *Talking / Conversation instead of Silence / Ambient Noise*	 Me Unavailable Since 3:56:15 PM When would... Sound be Talking / Conversation ? Volume 47 dB Periods of Silence 23.0 % Pitch (Frequency) Range Low Pitch High Pitch	 Me Unavailable Since 3:56:15 PM What else can Sound be ? *Talking / Conversation* (100.0% confidence) or *Listening to Music* (0.0% confidence) or *Silence / Ambient Noise* (0.0% confidence)	 Me Unavailable Since 3:56:15 PM What details affect Sound ? Volume 49 dB Periods of Silence 40.0 % Pitch (Frequency) Range Low Pitch High Pitch 1482Hz (from 1140Hz to 2623Hz) Pitch Pureness / Noisiness Pure 0 Noisiest 100 47 Pitch Fluctuation 1.14 Watts	 Periods of Silence Indicates the percentage of time that was quiet relative to the average. Speech has significant periods of silence, and so would have a higher value. Ok Periods of Silence 23.0 % Pitch (Frequency) Range Low Pitch High Pitch

G.1.4 Motion

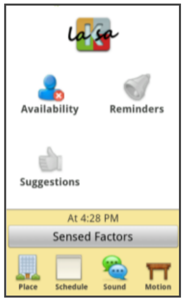
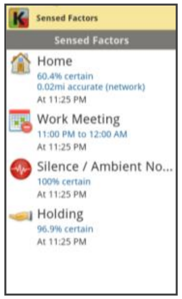
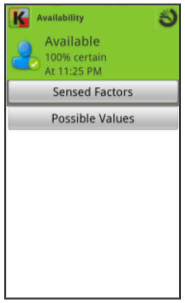
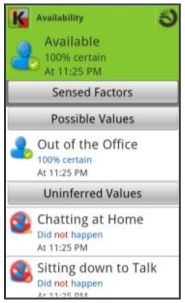
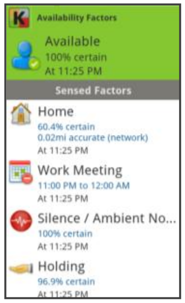
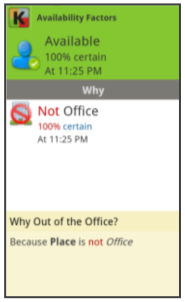
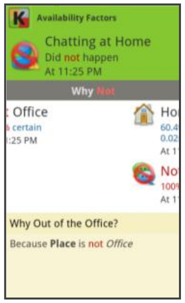
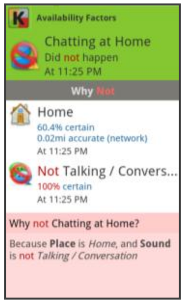
What	When	Why	Why Not	How Can	Outputs	Inputs
 Me Semi-Available Since 09:38:55 PM What is the Motion ? Motion would be Sitting with 0.0% confidence	 Me Semi-Available Since 09:38:55 PM When did Motion become Sitting ? Motion has been Sitting since 11:44:15 PM	 Me Semi-Available Since 09:38:55 PM Why is Motion Sitting ? Because -0.31g Force Z ≤ 0.15g Indicates how much force the phone sensed in the z direction. If the phone is lying on the back, then it would sense a force of +1g. and Force X ≤ 0.72g Indicates how much force the phone sensed in the x direction. If the phone is lying on the left side, then it would sense a force of +1g. and Power in Y ≤ 143.91Watts Indicates how energetic the movement in the y direction was.	 Me Semi-Available Since 09:38:55 PM Why isn't Motion Walking ? Because: Orientation YZ ≤ -57.8° Indicates the orientation about the x-axis. Zero value indicates the phone is lying on its back. The more positive the value, the more it rotates clockwise. and Force X ≤ -0.62g Indicates how much force the phone sensed in the x direction. If the phone is lying on the left side, then it would sense a force of +1g. and Power in Y ≤ 143.91Watts Indicates how energetic the movement in the y direction was.	 Me Semi-Available Since 09:38:55 PM How can Motion be Standing ? Reason 1 Reason 2 Power in Y ≤ 143.91Watts Indicates how energetic the movement in the y direction was. and Force Z ≤ 0.90g Indicates how much force the phone sensed in the z direction. If the phone is lying on the back, then it would sense a force of +1g. and Force Z ≤ 0.15g Indicates how much force the phone sensed in the z direction. If the phone is lying on the back, then it would sense a force of +1g. and Force Z > -0.31g Indicates how much force the phone sensed in the z direction. If the phone is lying on the back, then it would sense a force of +1g. and Force X ≤ 0.72g Indicates how much force the phone sensed in the x direction. If the phone is lying on the left side, then it would sense a force of +1g.	 Me Semi-Available Since 09:38:55 PM What values can Motion be ? Sitting or Standing or Walking or Running or Cycling or Vehicle or Holding or Other	 Me Semi-Available Since 09:38:55 PM What factors affects Motion ? Forces (Front View) Forces (Side View) Forces (Bottom View) Orientation (Front View) Orientation (Side View) Orientation (Bottom View) Force X (-0.99g) Indicates how much force the phone sensed in the x direction. If the phone is lying on the left side, then it would sense a force of +1g. Force Y (0.13g) Indicates how much force the phone sensed in the y direction. If the phone is lying on the bottom side, then it would sense a force of +1g. Force Z (-0.23g) Indicates how much force the phone sensed in the z direction. If the phone is lying on the back, then it would sense a force of +1g. Orientation XY (82.6°) Indicates the orientation about the z-axis. Zero value indicates the phone is lying on the left side. The more positive the value, the more it rotates clockwise.

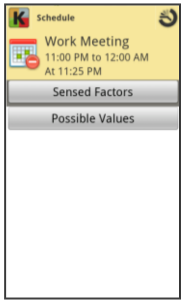
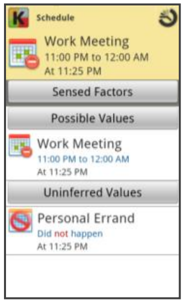
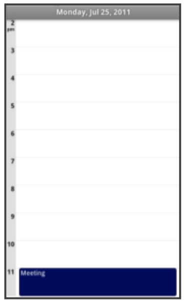
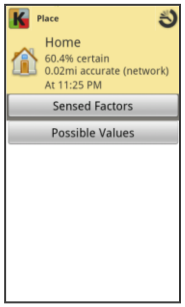
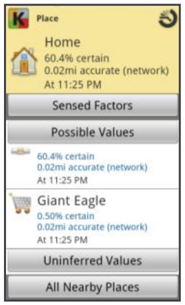
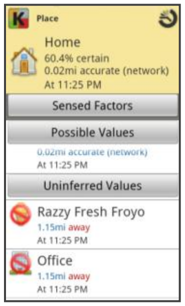
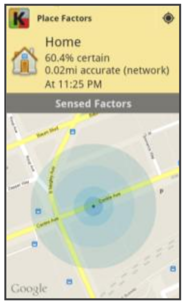
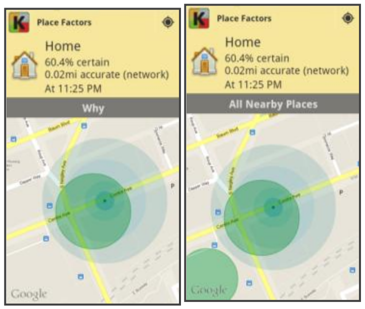
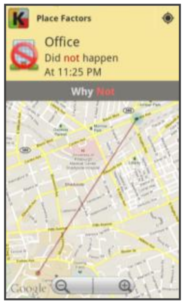
H Laκsa2 User Interface

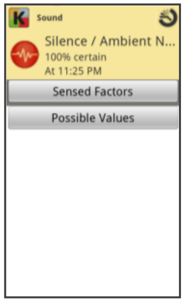
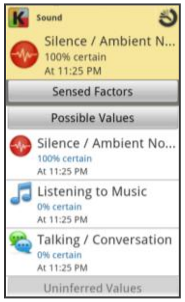
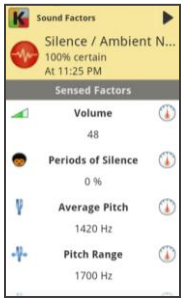
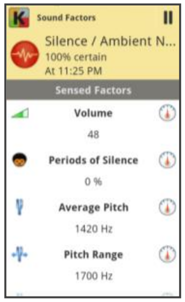
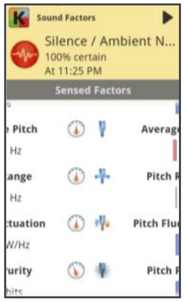
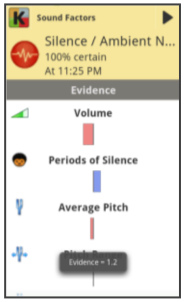
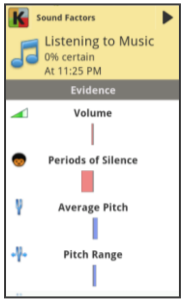
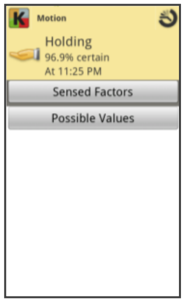
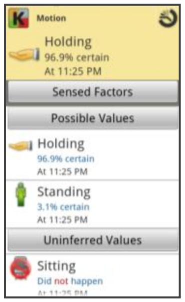
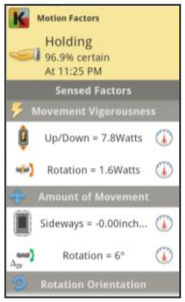
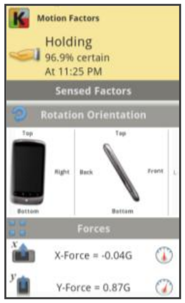
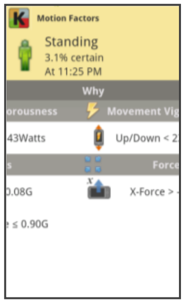
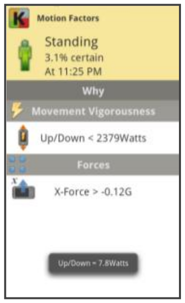
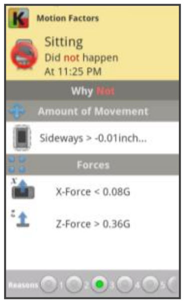
This appendix describes the user interface of various aspects of the Laκsa2 prototype used in the user study in Chapter 9. Only the Intelligibility User Interface was seen by participants, while the other interfaces were for administrative and configuration purposes. Participants also did not use the Motion inference and explanation features.

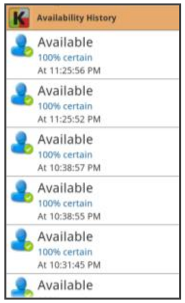
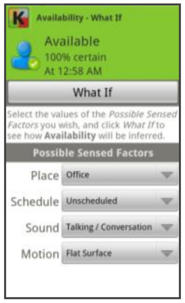
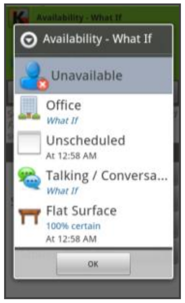
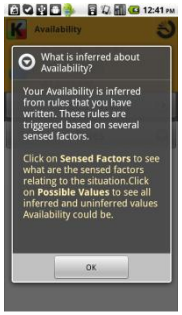
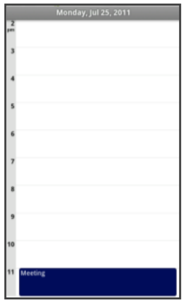
Furthermore, while Laκsa2 supports three services — Availability, Reminders, and Suggestions, participants were only exposed to the Availability service in the user study.

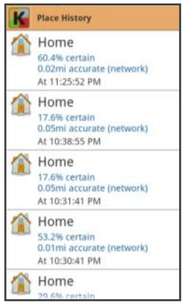
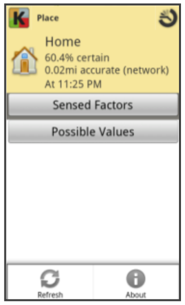
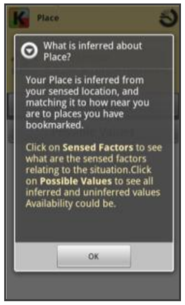
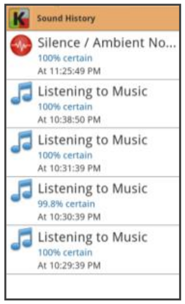
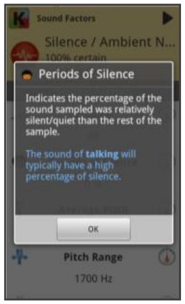
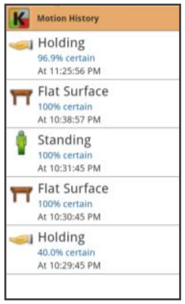
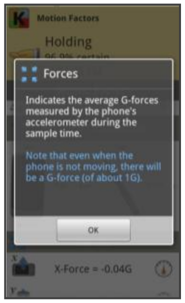
H.1 INTELLIGIBILITY USER INTERFACE

	What + Certainty	Outputs + Certainties	Inputs + Certainties + When	Why	Why Not
Home	 Availability Reminders Suggestions At 4:28 PM Sensed Factors Place Schedule Sound Motion Action states and sensed state		 Sensed Factors Sensed Factors Home 60.4% certain 0.02mi accurate (network) At 11:25 PM Work Meeting 11:00 PM to 12:00 AM At 11:25 PM Silence / Ambient No... 100% certain At 11:25 PM Holding 96.9% certain At 11:25 PM Showing sensed Inputs with Certainties and When.		
Availability	 Availability Available 100% certain At 11:25 PM Sensed Factors Possible Values	 Availability Available 100% certain At 11:25 PM Sensed Factors Possible Values Out of the Office 100% certain At 11:25 PM Uninferred Values Chatting at Home Did not happen At 11:25 PM Sitting down to Talk Did not happen At 11:25 PM Lists Output values. Requires explicit user interaction to see Uninferred values.	 Availability Factors Available 100% certain At 11:25 PM Sensed Factors Home 60.4% certain 0.02mi accurate (network) At 11:25 PM Work Meeting 11:00 PM to 12:00 AM At 11:25 PM Silence / Ambient No... 100% certain At 11:25 PM Holding 96.9% certain At 11:25 PM Similar to Inputs of home, but contextualizes for action (in this case, Availability)	 Availability Factors Available 100% certain At 11:25 PM Why Not Office 100% certain At 11:25 PM Why Out of the Office? Because **Place** is not *Office*  Availability Factors Chatting at Home Did not happen At 11:25 PM Why Not Office certain :25 PM Ho 60.4 0.02 At 1 No 100 At 1 Why Out of the Office? Because **Place** is not *Office* Filtered explanation only showing relevant factors. Textual natural language explanation for improved readability. *Flingable* to other reasons to support quick comparisons.	 Availability Factors Chatting at Home Did not happen At 11:25 PM Why Not Home 60.4% certain 0.02mi accurate (network) At 11:25 PM Not Talking / Convers... 100% certain At 11:25 PM Why not Chatting at Home? Because **Place** is *Home*, and **Sound** is not *Talking / Conversation*

	What + Certainty	Outputs + Certainties	Inputs + Certainties + When	Why	Why Not
Schedule	 Schedule Work Meeting 11:00 PM to 12:00 AM At 11:25 PM Sensed Factors Possible Values	 Schedule Work Meeting 11:00 PM to 12:00 AM At 11:25 PM Sensed Factors Possible Values Work Meeting 11:00 PM to 12:00 AM At 11:25 PM Uninferred Values Personal Errand Did not happen At 11:25 PM	 Monday, Jul 25, 2011 Meeting Show native Android calendar app as explanation of schedule		
Place	 Place Home 60.4% certain 0.02mi accurate (network) At 11:25 PM Sensed Factors Possible Values Certainty also described with location radius accuracy, and source (network/GPS)	 Place Home 60.4% certain 0.02mi accurate (network) At 11:25 PM Sensed Factors Possible Values 60.4% certain 0.02mi accurate (network) At 11:25 PM Giant Eagle 0.50% certain 0.02mi accurate (network) At 11:25 PM Uninferred Values All Nearby Places  Place Home 60.4% certain 0.02mi accurate (network) At 11:25 PM Sensed Factors Possible Values 0.02mi accurate (network) At 11:25 PM Uninferred Values Razzy Fresh Froyo 1.15mi away At 11:25 PM Office 1.15mi away At 11:25 PM	 Place Factors Home 60.4% certain 0.02mi accurate (network) At 11:25 PM Sensed Factors Input coordinates represented by center dot. Concentric circles indicating accuracy bounds for: 15%, 50%, 95%, 99.9% CEP (circular error probable)	 Place Factors Home 60.4% certain 0.02mi accurate (network) At 11:25 PM Why Place Factors Home 60.4% certain 0.02mi accurate (network) At 11:25 PM All Nearby Places Area overlap with selected Place to ask Why. *Higher bandwidth*: can also show overlaps with multiple places (including nearby uninferred Places).	 Place Factors Office Did not happen At 11:25 PM Why Not Also draws line from uninferred Place to user Location.

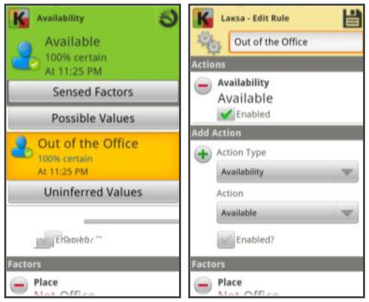
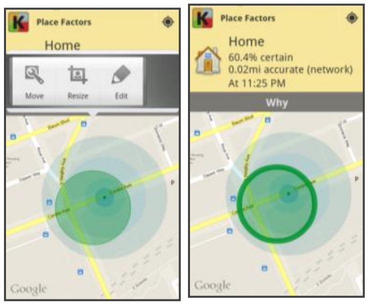
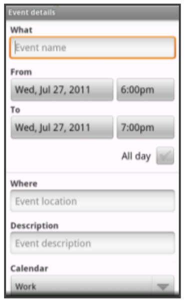
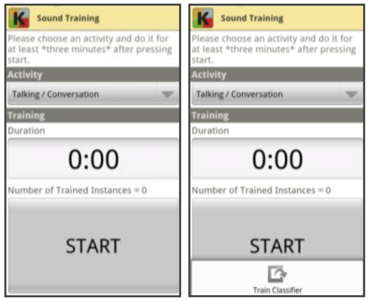
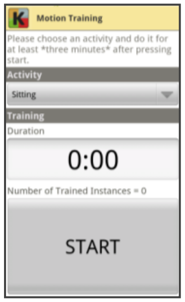
	What + Certainty	Outputs + Certainties	Inputs + Certainties + When	Why	Why Not
Sound	 Sound Silence / Ambient N... 100% certain At 11:25 PM Sensed Factors Possible Values	 Sound Silence / Ambient N... 100% certain At 11:25 PM Sensed Factors Possible Values Silence / Ambient No... 100% certain At 11:25 PM Listening to Music 0% certain At 11:25 PM Talking / Conversation 0% certain At 11:25 PM Uninferred Values	 Sound Factors Silence / Ambient N... 100% certain At 11:25 PM Sensed Factors Volume 48 Periods of Silence 0 % Average Pitch 1420 Hz Pitch Range 1700 Hz  Sound Factors Silence / Ambient N... 100% certain At 11:25 PM Sensed Factors Volume 48 Periods of Silence 0 % Average Pitch 1420 Hz Pitch Range 1700 Hz Numeric values indicating sensed value of features. Gauges indicating whether value is relatively high/low/average. *Flingable* to support quick comparisons.	 Sound Factors Silence / Ambient N... 100% certain At 11:25 PM Sensed Factors Pitch Averag Hz ange Pitch R Hz tuation Pitch Flu W/Hz urity Pitch F bits  Sound Factors Silence / Ambient N... 100% certain At 11:25 PM Evidence Volume Periods of Silence Average Pitch Evidence = 1.2 Bar chart visualizations indicating weights of evidences.	 Sound Factors Listening to Music 0% certain At 11:25 PM Evidence Volume Periods of Silence Average Pitch Pitch Range
Motion	 Motion Holding 96.9% certain At 11:25 PM Sensed Factors Possible Values	 Motion Holding 96.9% certain At 11:25 PM Sensed Factors Possible Values Holding 96.9% certain At 11:25 PM Standing 3.1% certain At 11:25 PM Uninferred Values Sitting Did not happen At 11:25 PM	 Motion Factors Holding 96.9% certain At 11:25 PM Sensed Factors Movement Vigorousness Up/Down = 7.8Watts Rotation = 1.6Watts Amount of Movement Sideways = -0.00inch... Rotation = 6° Rotation Orientation  Motion Factors Holding 96.9% certain At 11:25 PM Sensed Factors Rotation Orientation Top Right Bottom Top Back Front Bottom Forces X-Force = -0.04G Y-Force = 0.87G Similar to Sound, but also includes physical diagram to show Orientation.	 Motion Factors Standing 3.1% certain At 11:25 PM Why orousness Movement Vig 43Watts Up/Down < 2 s Force 0.08G X-Force > - ≤ 0.90G  Motion Factors Standing 3.1% certain At 11:25 PM Why Movement Vigorousness Up/Down < 2379Watts Forces X-Force > -0.12G Up/Down = 7.8Watts	 Motion Factors Sitting Did not happen At 11:25 PM Why Not Amount of Movement Sideways > -0.01inch... Forces X-Force < 0.08G Z-Force > 0.36G Reasons ○ 1 ○ 2 ● 3 ○ 4 ○ 5 ○ Multiple reasons via radio choice.

	History	What If	Description / Terminological
Availability	 Availability History Available 100% certain At 11:25:56 PM Available 100% certain At 11:25:52 PM Available 100% certain At 10:38:57 PM Available 100% certain At 10:38:55 PM Available 100% certain At 10:31:45 PM Available User can access historical instances by selecting from list, to go to What explanation of time.	 Availability - What If Available 100% certain At 12:58 AM What If Select the values of the *Possible Sensed Factors* you wish, and click *What If* to see how **Availability** will be inferred. Possible Sensed Factors Place: Office Schedule: Unscheduled Sound: Talking / Conversation Motion: Flat Surface  Availability - What If Unavailable Office *What If* Unscheduled At 12:58 AM Talking / Conversa... *What If* Flat Surface 100% certain At 12:58 AM OK	 12:41 PM Availability What is inferred about Availability? Your Availability is inferred from rules that you have written. These rules are triggered based on several sensed factors. Click on **Sensed Factors** to see what are the sensed factors relating to the situation.Click on **Possible Values** to see all inferred and uninferred values Availability could be. OK Accessed via the context menu's About menu item.
Schedule	 Monday, Jul 25, 2011 Meeting Show native Android calendar app as explanation of schedule		

	History	What If	Description / Terminological
Place	 Place History Home 60.4% certain 0.02mi accurate (network) At 11:25:52 PM Home 17.6% certain 0.05mi accurate (network) At 10:38:55 PM Home 17.6% certain 0.05mi accurate (network) At 10:31:41 PM Home 53.2% certain 0.01mi accurate (network) At 10:30:41 PM Home 29.6% certain		 Place Home 60.4% certain 0.02mi accurate (network) At 11:25 PM Sensed Factors Possible Values Refresh About  Place What is inferred about Place? Your Place is inferred from your sensed location, and matching it to how near you are to places you have bookmarked. Click on **Sensed Factors** to see what are the sensed factors relating to the situation.Click on **Possible Values** to see all inferred and uninferred values Availability could be. OK
Sound	 Sound History Silence / Ambient No... 100% certain At 11:25:49 PM Listening to Music 100% certain At 10:38:50 PM Listening to Music 100% certain At 10:31:39 PM Listening to Music 99.8% certain At 10:30:39 PM Listening to Music 100% certain At 10:29:39 PM		 Sound Factors Silence / Ambient N... 100% certain Periods of Silence Indicates the percentage of the sound sampled was relatively silent/quiet than the rest of the sample. The sound of **talking** will typically have a high percentage of silence. OK Pitch Range 1700 Hz Terminological explanations when clicking on labels. Descriptive explanations via Options menu.
Motion	 Motion History Holding 96.9% certain At 11:25:56 PM Flat Surface 100% certain At 10:38:57 PM Standing 100% certain At 10:31:45 PM Flat Surface 100% certain At 10:30:45 PM Holding 40.0% certain At 10:29:45 PM		 Motion Factors Holding 96.9% certain Forces Indicates the average G-forces measured by the phone's accelerometer during the sample time. Note that even when the phone is not moving, there will be a G-force (of about 1G). OK X-Force = -0.04G Similar to Sound

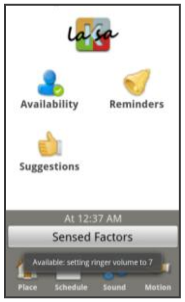
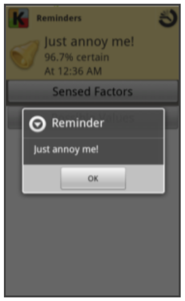
H.2 Control User Interfaces

This UI supports setting up rules and to train machine learning models for the experiment. It was only used by the experimenter.

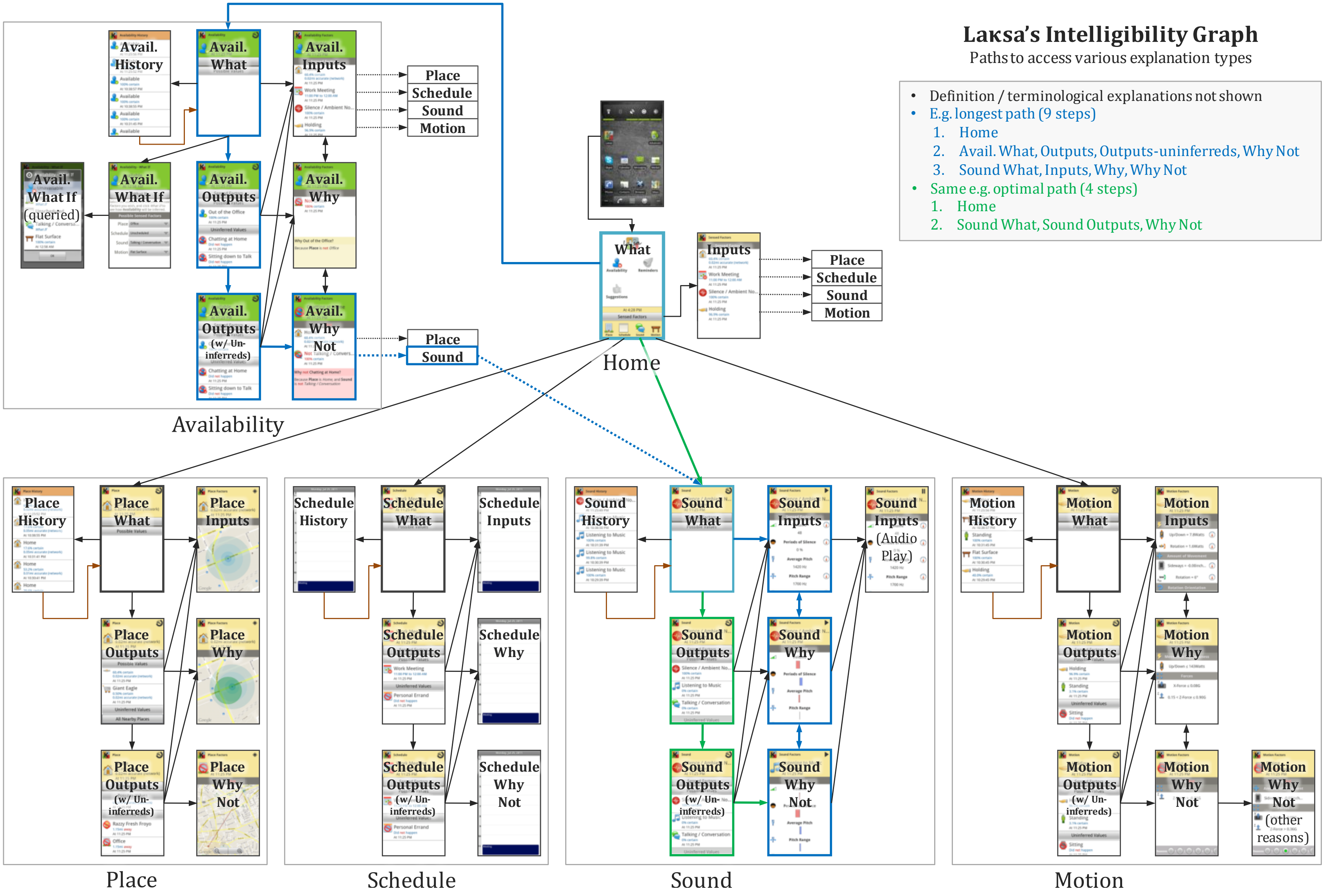
Rules	Place	Schedule	Sound	Motion
 Availability Available 100% certain At 11:25 PM Sensed Factors Possible Values Out of the Office 100% certain At 11:25 PM Uninferred Values Factors Place Laksa - Edit Rule Out of the Office Actions Availability Available Enabled Add Action Action Type Availability Action Available Enabled? Factors Place Edit accessed via long-press of rules anywhere they are seen. Can add new rule via Options Menu.	 Place Factors Home Move Resize Edit Google Place Factors Home 60.4% certain 0.02mi accurate (network) At 11:25 PM Why Google Accessed by long-tapping Place bubble (green or red). Can move or resize.	 Event details What Event name From Wed, Jul 27, 2011 6:00pm To Wed, Jul 27, 2011 7:00pm All day Where Event location Description Event description Calendar Work Add and edit via native Android calendar app.	 Sound Training Please choose an activity and do it for at least *three minutes* after pressing start. Activity Talking / Conversation Training Duration 0:00 Number of Trained Instances = 0 START Sound Training Please choose an activity and do it for at least *three minutes* after pressing start. Activity Talking / Conversation Training Duration 0:00 Number of Trained Instances = 0 START Train Classifier Training for Sound instances. Can train classifier from saved instances.	 Motion Training Please choose an activity and do it for at least *three minutes* after pressing start. Activity Sitting Training Duration 0:00 Number of Trained Instances = 0 START Training for Motion instances. Can train classifier from saved instances.

H.3 ACTIONS USER INTERFACE

As part of the original range of features, Laκsa2 supported three Actions: setting Availability, triggering Reminders, and triggering third-party-set Suggestions.

Availability	Reminders	Suggestions
 Availability Reminders Suggestions Sensed Factors Place Schedule Sound Motion At 12:37 AM Available: setting ringer volume to 7 Ringer mode changes Available: Normal Semi-Available: Vibrate Unavailable: Silent	 Just annoy me! 96.7% certain At 12:36 AM Just annoy me! Laκsa wakes the screen, unlocks key-guard, brings up Reminder page, and pop-ups Alert Dialog with reminder message when contextual situation is met.	Similar to Reminders, but rules will have been set by others (commercial advertisers, social network, *etc.*), and not by user.

H.4 LAKSA INTELLIGIBILITY GRAPH



Laκsa's Intelligibility Graph

Paths to access various explanation types

- Definition / terminological explanations not shown
- E.g. longest path (9 steps)
  1. Home
  2. Avail. What, Outputs, Outputs-uninferreds, Why Not
  3. Sound What, Inputs, Why, Why Not
- Same e.g. optimal path (4 steps)
  1. Home
  2. Sound What, Sound Outputs, Why Not

I LAKSA2 STUDY MATERIALS

This appendix details several instruction and scenario materials presented to participants in the study presented in Chapter 9.

I.1 INSTRUCTIONS

Instructions that participants read before interacting with the Laκsa prototype. Participants may ask the experimenter for clarifications.

User Study Instructions

Your Objective

Imagine you work in an office with a team of coworkers. The scenarios we will cover pertain to this job and a particular project you are currently working on.

You also recently installed a new smart phone application, called Laκsa, to help *automatically* set your availability based on what it can sense about you. This can help improve the productivity of your team by indicating when to contact one another, and reducing unnecessary interruptions. While it is often correct, it sometimes makes mistakes (as does everyone). However, it can **explain** to you what it knows and what it is thinking.

You will evaluate the performance of Laκsa to determine when and how best to use it, *i.e.*, when it works well, and when it makes mistakes. Try to learn enough about Laκsa to be able to **teach** them how to use it, and **configure** it so that it best performs for them and yourself.

Find out whether the application performs **appropriately**, and if **not**, how to ***improve*** its performance, so that it does not make the mistakes again. You may make **changes** by:

1. Proposing to adjust some **settings** of the application
    a. Positions and sizes of places
    b. Calendar schedule
    c. Sound sensitivities
2. Proposing new **rules** to add to the application
3. Suggesting ways to change your **behavior** or **environment**

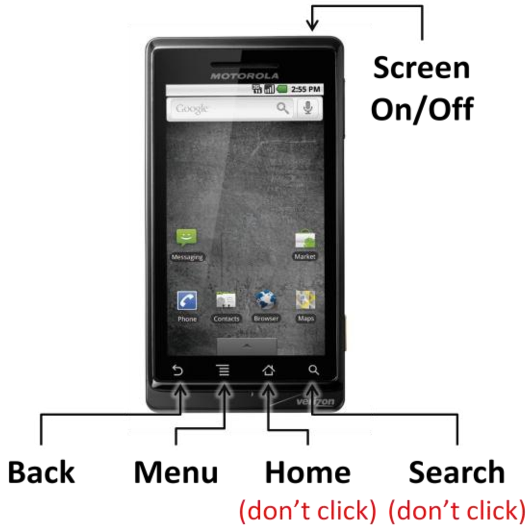
You may look at the application anytime you wish, even when nothing in particular is happening. As you use the application, feel free to tap on anything.

Scenarios

In this study, you will be going through several scenarios where you will be engaged in several activities. Laκsa will change your Availability status at various times. Even though the experimenter is nearby, as long as the experimenter does not talk to you, imagine he is not around.

Android Phone Instructions

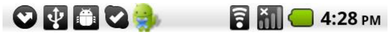
The following diagram shows you buttons on the phone that you will need to use during the study.



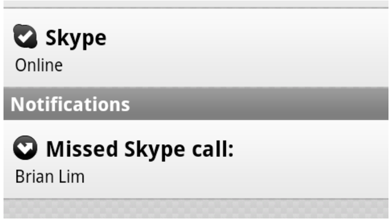
Skype Instructions

All calls in this study will be done through Skype. You may receive calls, or miss them. If you miss a call, you will see the

icon on the top notification bar as such:



You can drag the top bard down to see this:

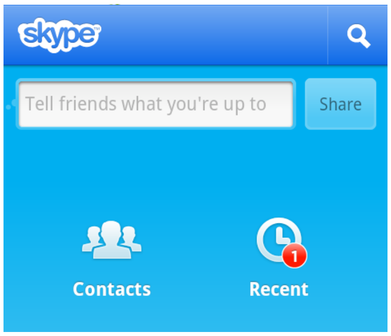


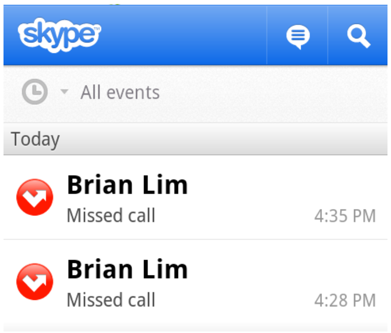
Skype
Online

Notifications

Missed Skype call:
Brian Lim

Clicking on the Skype section, will bring you to the Skype application, where you can see any recent events, and at what times they happened.





All events

Today

Brian Lim
Missed call 4:35 PM

Brian Lim
Missed call 4:28 PM

Laκsa Application Instructions

Laκsa is a smart phone application that senses what is happening to proactively change your availability and send you appropriate reminders. For example, it can set your status to Unavailable if it detects that you are in the Office, and are involved in a Conversation.

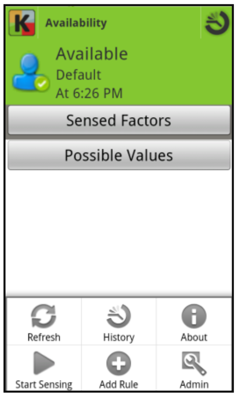
It uses sophisticated *sensing* and *inference* to figure out what you are doing. While it is often correct, it sometimes makes mistakes (as does everyone). However, it can **explain** to you what it knows and what it is thinking.

Availability

There are times when you would rather not be interrupted when you are busy, such as when in a meeting, or working frantically in the office. Using ***rules*** that have been written, Laκsa can automatically set your availability, and the phone's ringer mode. Hence, you will not be disturbed when you should not be, and you will not forget to set your ringer to normal when you become available again.

Your availability may be *automatically* set to: **Available**, **Semi-Available**, or **Unavailable**.

These correspond to ringer modes: **Normal**, **Vibrate**, **Silent**, respectively.

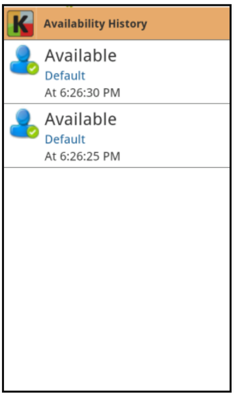


Availability

Available
Default
At 6:26 PM

Sensed Factors

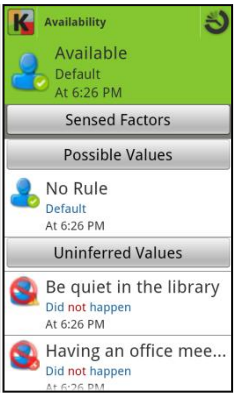
Possible Values



Availability History

Available
Default
At 6:26:30 PM

Available
Default
At 6:26:25 PM



Availability

Available
Default
At 6:26 PM

Sensed Factors

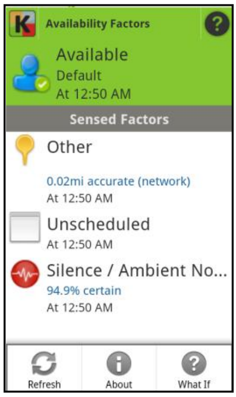
Possible Values

No Rule
Default
At 6:26 PM

Uninferred Values

Be quiet in the library
Did not happen
At 6:26 PM

Having an office mee...
Did not happen
At 6:26 PM



Availability Factors

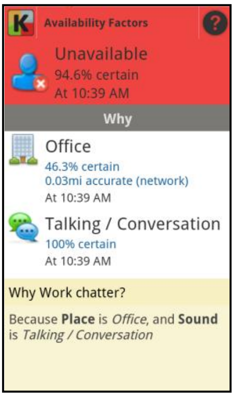
Available
Default
At 12:50 AM

Sensed Factors

Other
0.02mi accurate (network)
At 12:50 AM

Unscheduled
At 12:50 AM

Silence / Ambient No...
94.9% certain
At 12:50 AM



Availability Factors

Unavailable
94.6% certain
At 10:39 AM

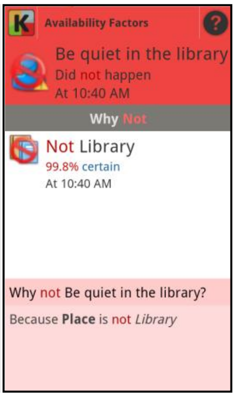
Why

Office
46.3% certain
0.03mi accurate (network)
At 10:39 AM

Talking / Conversation
100% certain
At 10:39 AM

Why Work chatter?

Because **Place** is *Office*, and **Sound** is *Talking / Conversation*



Availability Factors

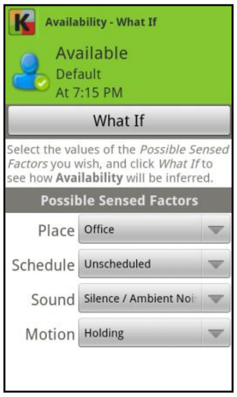
Be quiet in the library
Did not happen
At 10:40 AM

Why Not

Not Library
99.8% certain
At 10:40 AM

Why not Be quiet in the library?

Because **Place** is not *Library*



Availability - What If

Available
Default
At 7:15 PM

What If

Select the values of the *Possible Sensed Factors* you wish, and click *What If* to see how **Availability** will be inferred.

Possible Sensed Factors

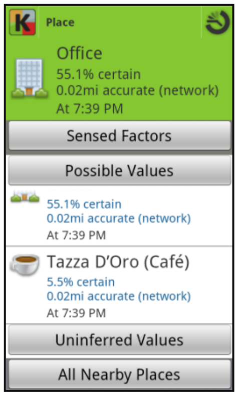
Place: Office
Schedule: Unscheduled
Sound: Silence / Ambient Noi
Motion: Holding

Sensed Factors

Laĸsa figures out what is happening by sensing several factors: Place, Schedule, and Sound.

Place

Laĸsa can detect your location and determine if you are near any places you have *previously* saved. For example, it can tell whether you are at Home, in the Office, or near the Grocery Store.



Place

Office
55.1% certain
0.02mi accurate (network)
At 7:39 PM

Sensed Factors

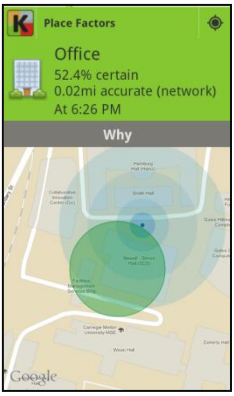
Possible Values

55.1% certain
0.02mi accurate (network)
At 7:39 PM

Tazza D'Oro (Café)
5.5% certain
0.02mi accurate (network)
At 7:39 PM

Uninferred Values

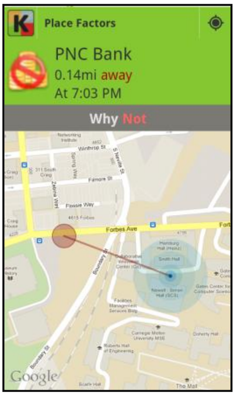
All Nearby Places



Place Factors

Office
52.4% certain
0.02mi accurate (network)
At 6:26 PM

Why



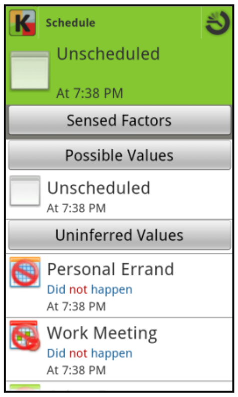
Place Factors

PNC Bank
0.14mi away
At 7:03 PM

Why Not

Schedule

Laĸsa can determine if you have any events scheduled on your Google calendar.



Schedule

Unscheduled
At 7:38 PM

Sensed Factors

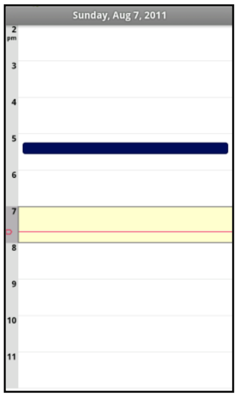
Possible Values

Unscheduled
At 7:38 PM

Uninferred Values

Personal Errand
Did not happen
At 7:38 PM

Work Meeting
Did not happen
At 7:38 PM

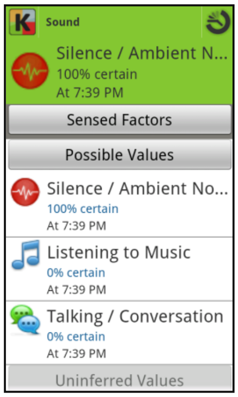


Sunday, Aug 7, 2011

Sound

Laĸsa can hear the sounds near you and recognize what you may be doing or listening to. It can distinguish between the following three sound activities:

1. **Talking / Conversation**,
2. **Listening to Music**,
3. **Silence / Ambient Noise**.



Sound

Silence / Ambient N...
100% certain
At 7:39 PM

Sensed Factors

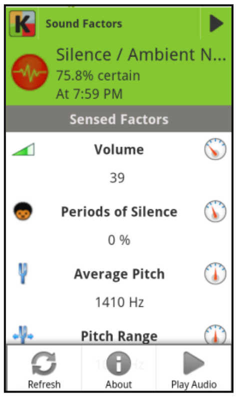
Possible Values

Silence / Ambient No...
100% certain
At 7:39 PM

Listening to Music
0% certain
At 7:39 PM

Talking / Conversation
0% certain
At 7:39 PM

Uninferred Values



Sound Factors

Silence / Ambient N...
75.8% certain
At 7:59 PM

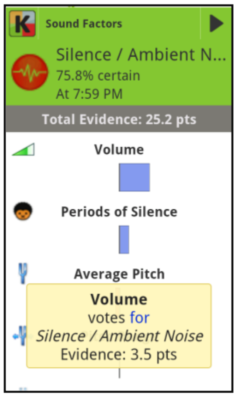
Sensed Factors

Volume
39

Periods of Silence
0 %

Average Pitch
1410 Hz

Pitch Range



Sound Factors

Silence / Ambient N...
75.8% certain
At 7:59 PM

Total Evidence: 25.2 pts

Volume

Periods of Silence

Average Pitch

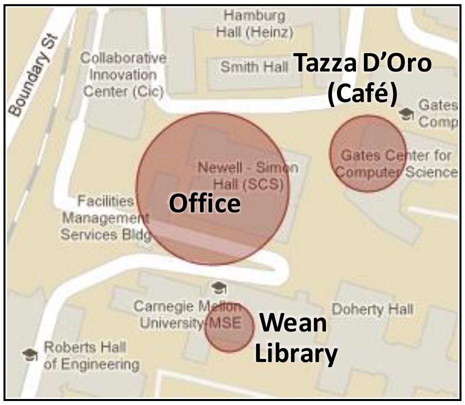
Volume
votes for
Silence / Ambient Noise
Evidence: 3.5 pts

Rules

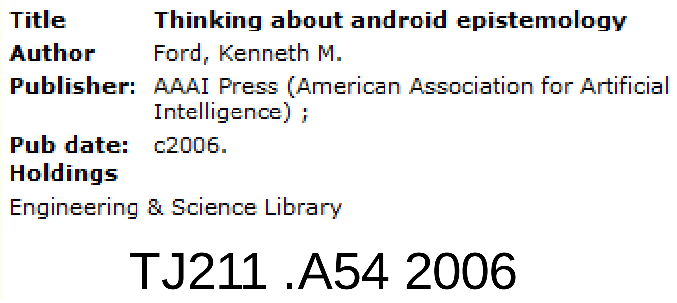
The following rules and saved places have been set up for you for the scenarios. Please study them.

				Sensed Factors		
#	Rule	Availability	Ringer	Place	Schedule	Sound
1	Having an office meeting	Unavailable	Silent	Office	Meeting	
2	Work chatter	Unavailable	Silent	Office		Talking
3	Someone's talking	Semi-Available	Vibrate			Talking
4	Be quiet in the library	Semi-Available	Vibrate	Hunt Library		
Default		Available	Ring	*All other cases*		

Saved Places


Boundary St
Hamburg Hall (Heinz)
Collaborative Innovation Center (Cic)
Smith Hall
Tazza D'Oro (Café)
Gates Comp
Gates Center for Computer Science
Newell - Simon Hall (SCS)
Office
Facilities Management Services Bldg
Carnegie Mellon University
MSE
Wean Library
Doherty Hall
Roberts Hall of Engineering

Library Book



Title **Thinking about android epistemology**
Author Ford, Kenneth M.
Publisher: AAAI Press (American Association for Artificial Intelligence) ;
Pub date: c2006.
Holdings
Engineering & Science Library

TJ211 .A54 2006

I.2 SCENARIOS

Checklist for experimenter to cover scenarios with participant with estimated times for each scenario.

#	Scenario / Critical Incident	Time	
		Plan	Actual
-	**Briefing / intro**	0:00	
	1. Sign IRB consent form 2. Read instructions; can keep for reference throughout study 3. Verify participant understands app functionality		
1	**Conversation in office**	0:10	
	1. Experimenter walks through interface (Scenario 0) 2. Verify participant understands app UI, by asking her to derive an explanation (e.g. why detected talking, why Unavailable)		
-	**Read news and play music (heavy vocals: Sound of Silence, 3 min)**	0:20	
	1. Experimenter tells participant it is OK to listen to music at the office, and asks her to play the song Sound of Silence by Simon and Garfunkel (http://www.youtube.com/watch?v=eZGWQauQOAQ). 2. Participant goes to http://www.cnn.com to read their favorite news.		
2	**Miss call from coworker because of Music**	0:30	
	1. Experimenter makes a call to phone; participant expected to miss call, since phone silent. 2. Call again after the music is over, and there is silence 3. *"Your coworker has been trying to call you for the past 3 minutes. He asks why you didn't pick up sooner."*		
-	**Walk to library to borrow books**	0:40	
	1. *"You need to borrow a book from a nearby library."* 2. Give participants call numbers and book titles. 3. Enter Wean library until sufficiently deep (to not disturb other patrons); have participant look for 1st book		
3	**Phone rings in Wean library**	0:45	
	1. Do not talk to get sound recognition as Silence. 2. Experimenter makes a call to phone; participant's phone will ring loudly. 3. *Try not to prompt: "Your friend is calling you. Remember we're in the library. How do you feel?"*		
4	**Investigate availability in Café**	0:55	
	1. *Prompt: "You want to ensure that you will be available to calls whenever you go for coffee breaks."* 2. Participant may or may not want / need to go to the actual café.		

Scenario 1

[You are talking to me now.]
Check if you will be interrupted with a phone call,
i.e., that your phone will **not ring** if someone calls you.

Also, **explore** the application as much as possible
to become familiar with it.

Scenario 2

Your coworker, **Damien**, is calling you, and says:

> *"Hello! I've been trying to call you for the past* ***3 minutes****. Could you hear your phone ring? Why didn't you pick up sooner?*
>
> *"Anyways, I wanted to check whether you've read my* ***email*** *yet. ... Please take a look at it as soon as possible and let me know what you think.*
>
> *"Also, please* ***get your phone fixed****, so you'll get my call next time."*

Scenario 3

Another coworker, **Evelyn**, is calling you, and says

> *“Hi! Is now a good time to call you?*
>
> *“I wanted to know if you have some time to spare? Can you help me with some figures for a document that I’m working on?”*

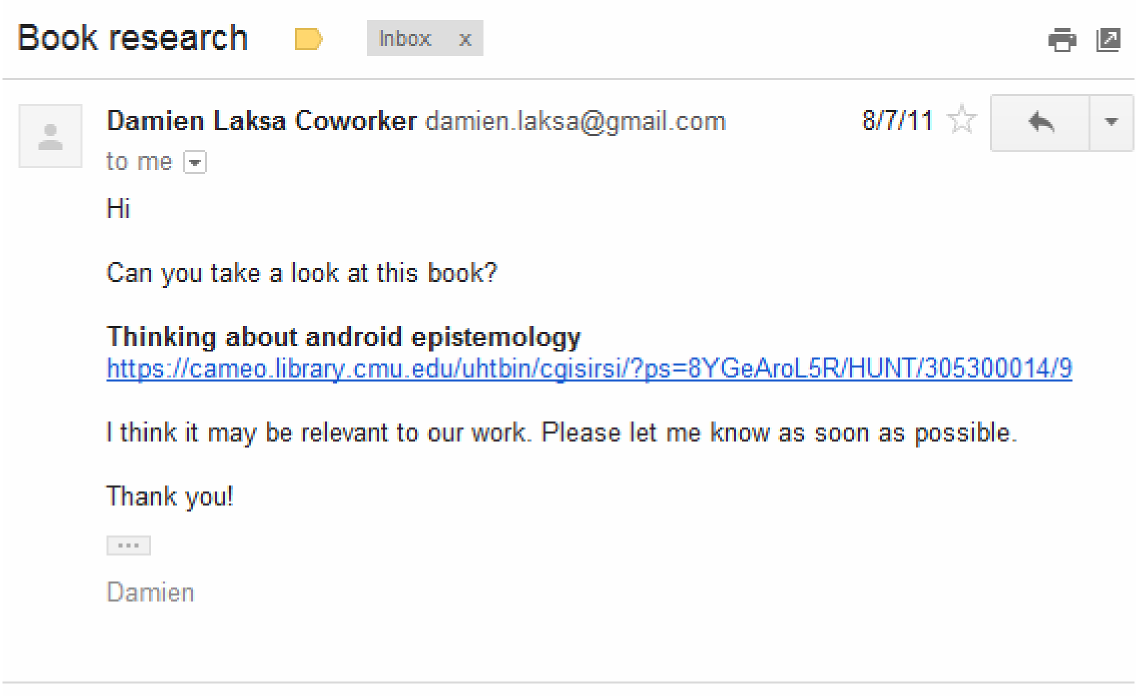
Scenario 4

You normally go to the nearby **Tazza D’Oro café** to get coffee or snacks during the day.

You want to check whether you will be able to **receive calls** if you are there.

[What will you do? Actually, do the actions, don’t just tell me.]

Scenario 2 Email



Book research

Damien Laksa Coworker damien.laksa@gmail.com 8/7/11

to me

Hi

Can you take a look at this book?

Thinking about android epistemology
https://cameo.library.cmu.edu/uhtbin/cgisirsi/?ps=8YGeAroL5R/HUNT/305300014/9

I think it may be relevant to our work. Please let me know as soon as possible.

Thank you!

Damien

I.3 Survey Instrument

Participant #: ________

Date: ____________________________

Start Time: ___________________ End Time: ___________________

DATABASE	
Backed up?	Uploaded?
AUDIO RECORDINGS	
Uploaded?	

Demographics

Gender: Male | Female Age: __________ Ethnicity: _______________

Education: _______________ Occupation: ______________________

Own a Smart Phone?

For how long? _______________

Which brand & model?

□ iPhone

□ Android

□ Windows 7 Mobile

□ Blackberry

Other: _____________________

Usage?

□ Web surfing

□ Email, Calendar

□ Play music

□ Camera

□ Social (Facebook, Twitter, *etc*)

□ Maps

□ Games

Other: _____________________

PLEASE DON'T TURN OVER

Questionnaire (Scenario 1)

Trigger Time: ______________

Notes:

START RECORDING AUDIO

Interview Time: ____________

1. Please rate whether you agree with the following statements

	Strongly Disagree	Disagree	Somewhat Disagree	Neither	Somewhat Agree	Agree	Strongly Agree
Laκsa **behaved appropriately** in this situation.							
It was **important** that Laκsa behaved appropriately in this situation.							
Laκsa's **behavior** was **useful** in this situation (correct/wrong)							
Laκsa's ***explanations*** were **helpful** for me to understand this situation							

2. How do you **feel** about what just happened?

3. What did you **want to know** in this situation?

4. What do you **understand** about what the application did and thought? **Why**?

To improve Laκsa's behavior, how will you change:

5. The **application settings**? **Rules**?

6. **Your behavior**?

J LAKSA2 USAGE DETAILS

We present some data of participant usage of Intelligibility features for the study in Chapter 9.

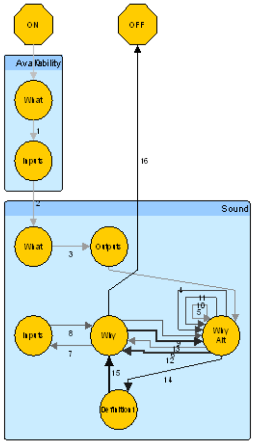
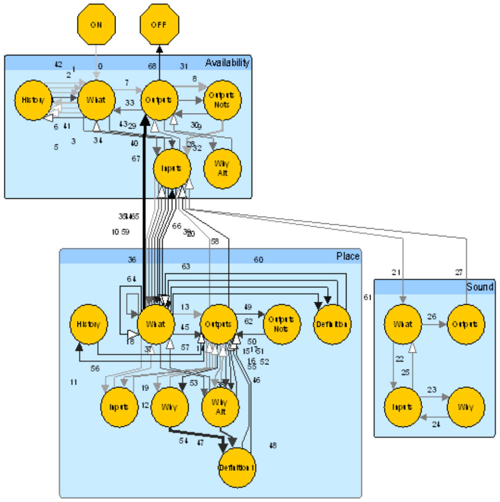
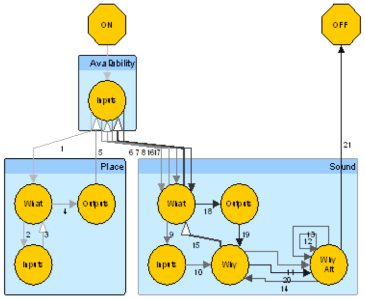
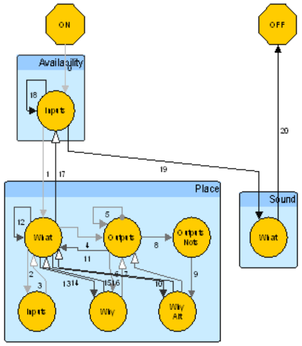
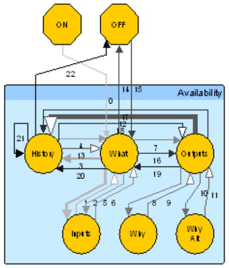
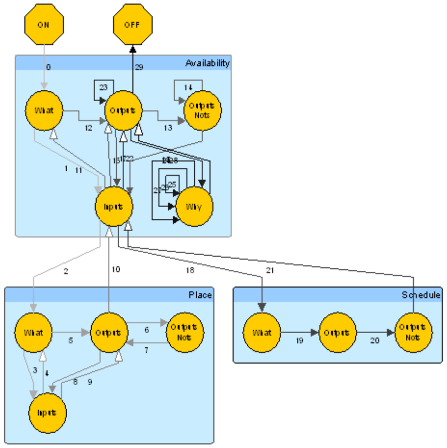
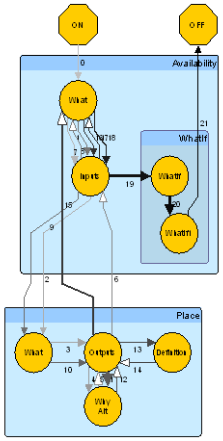
J.1 PARTICIPANT SEQUENCE MODELS

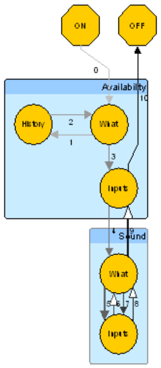
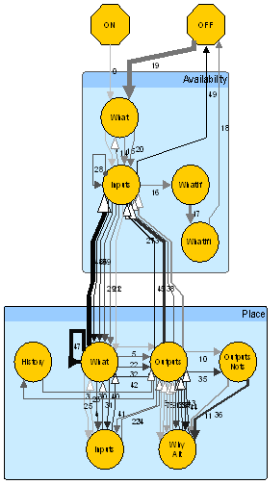
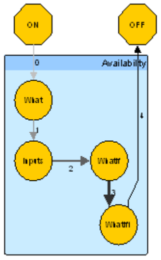
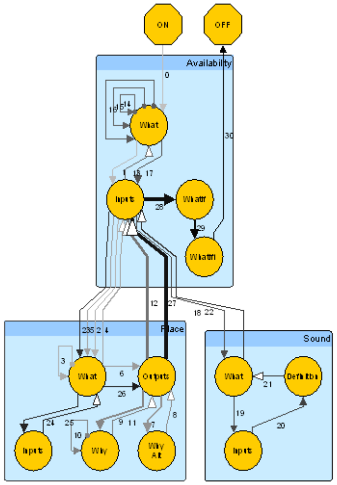
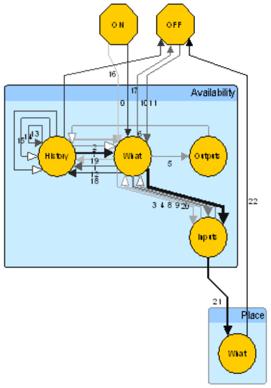
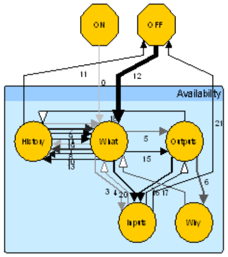
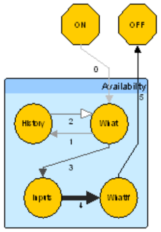
We extracted data logs from the Sqlite3 database of the Android smart phone (Motorola Droid) which participants used during the study. This was cleansed for occasional errors such as

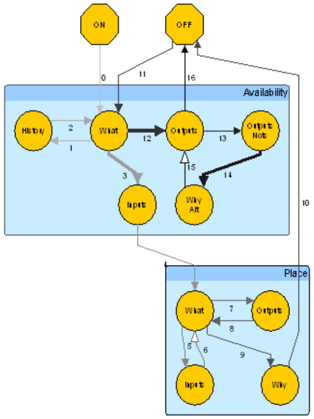
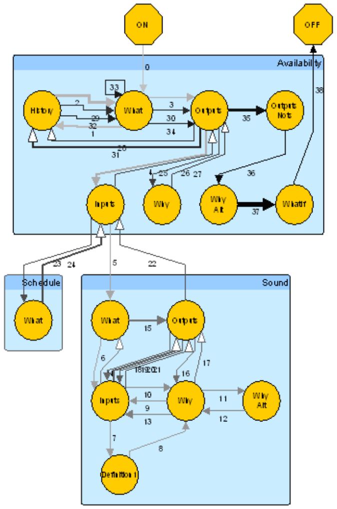
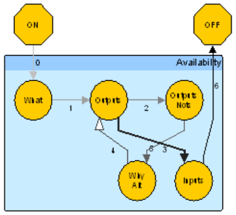
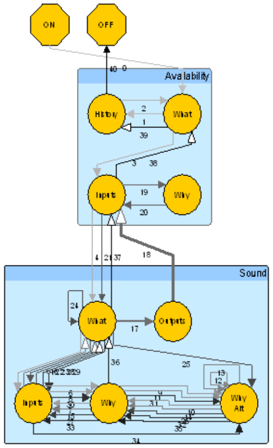
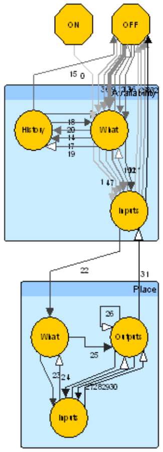
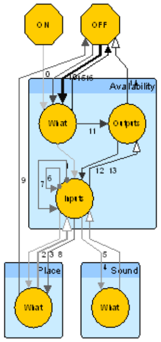
- User unintentional taps (identified by quickly tapping back within 0.5 sec)
- Repeated log entries due to lag

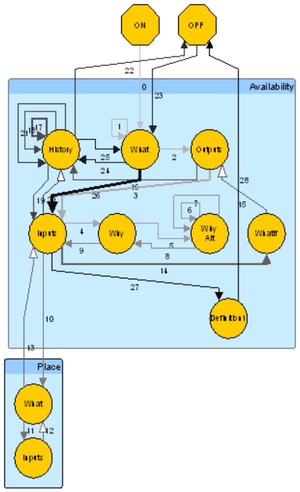
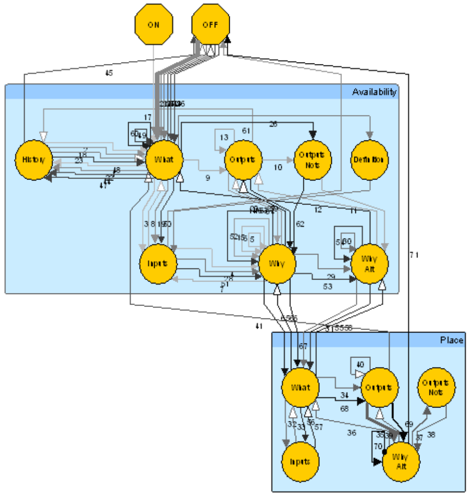
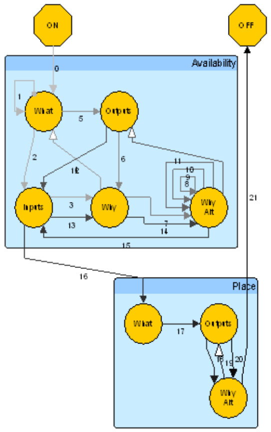
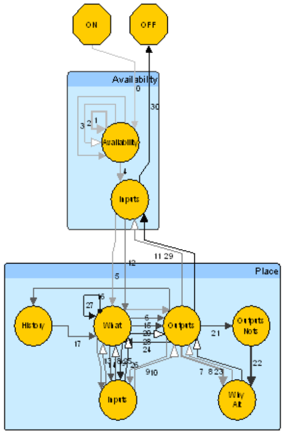
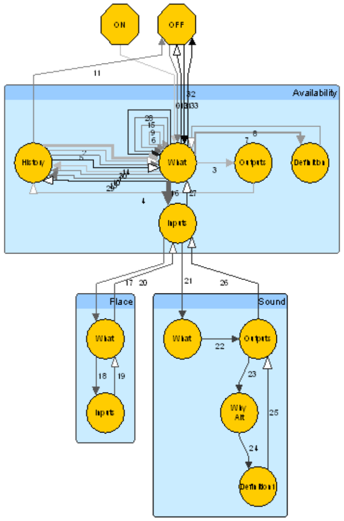
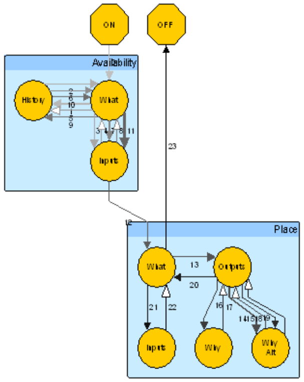
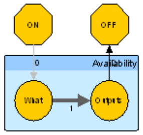
We parsed the logs into separate users and generated network graphs representing sequence diagrams for each participant during each scenario. We present these sequence diagrams in this section. How to read sequence diagrams:

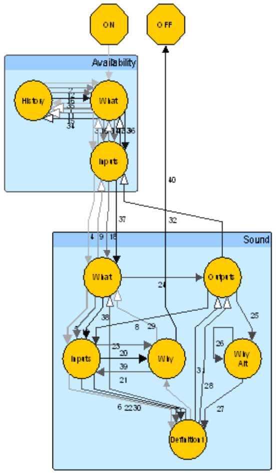
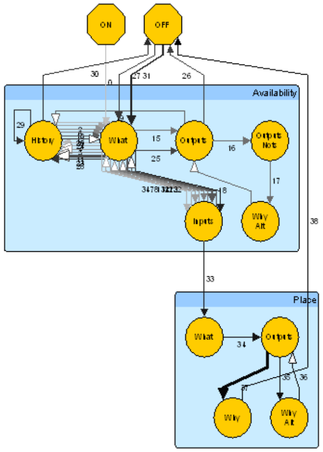
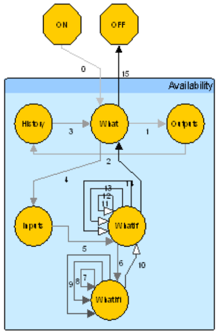
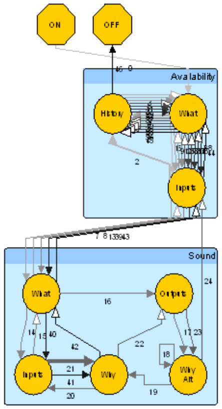
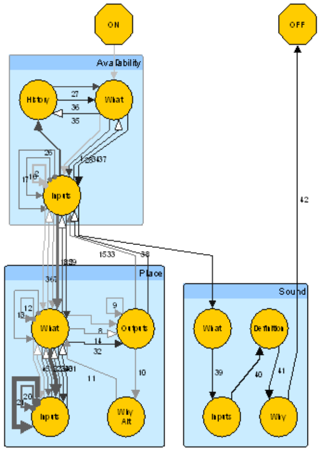
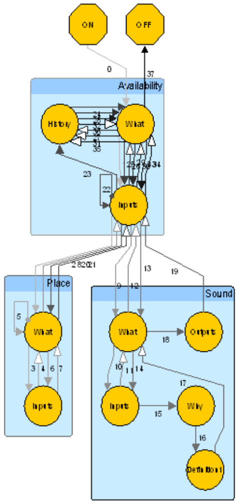
- **Rectangular Zone:** context type of explanation being viewed
- **Circular Node:** explanation type of the context type viewed
- **Hexagonal Node:** start (screen turned on) or stop (screen turned off)
- **Edge:** view transition from source page to destination page
    - **Thickness:** duration *source* (previous) explanation was viewed
    - **Number:** sequence step
    - **Shade:** corresponds to sequence step
    - **Solid Arrow:** user explicitly selected explanation
    - **Hollow Arrow:** user selected ‘Back’
    - **Cyclic Loop:** user selected ‘Refresh’

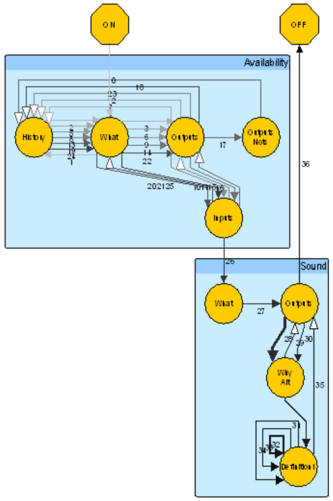
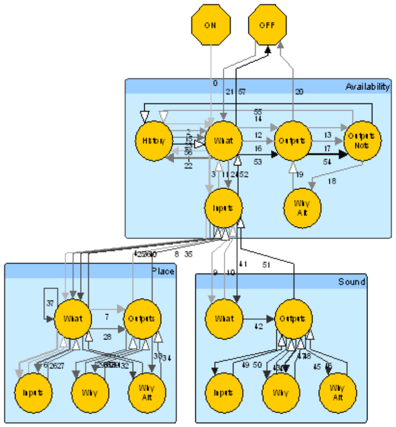
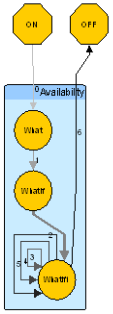
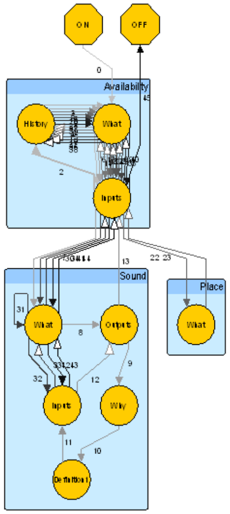
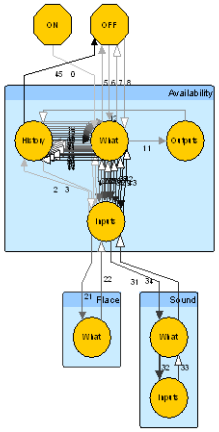
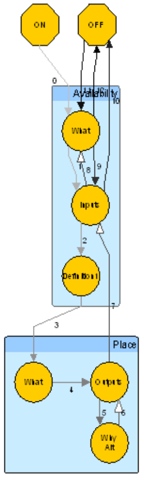
P#	S2	S3	S4-if	S4-situ
01				
02				

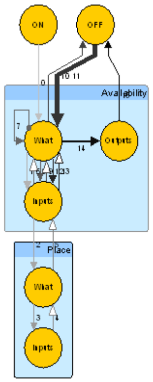
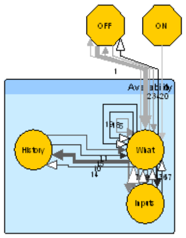
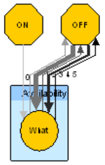
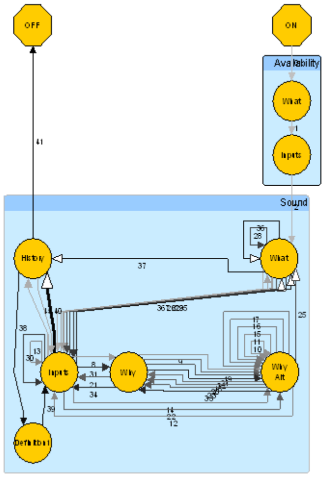
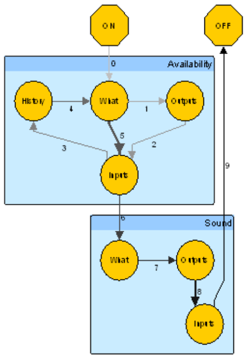
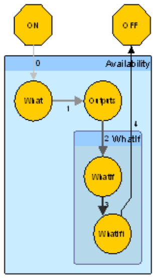
P#	S2	S3	S4-if	S4-situ
03				
04				

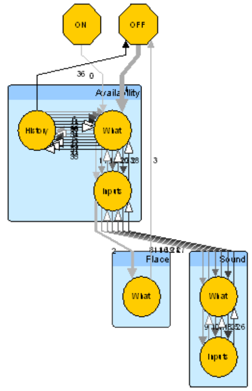
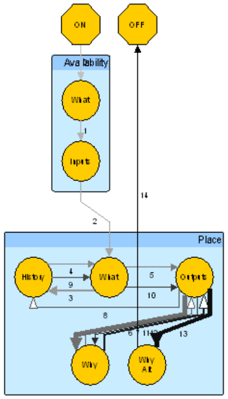
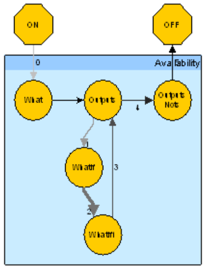
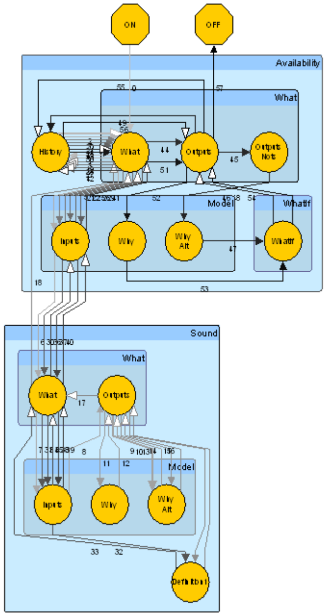
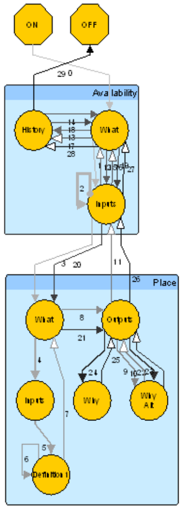
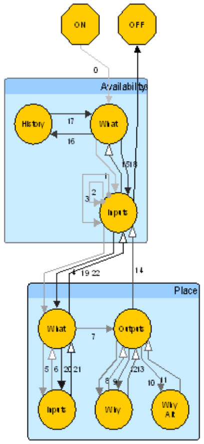
P#	S2	S3	S4-if	S4-situ
05				
06				

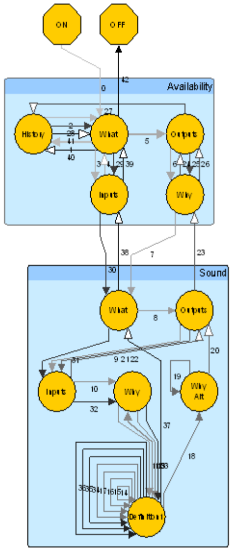
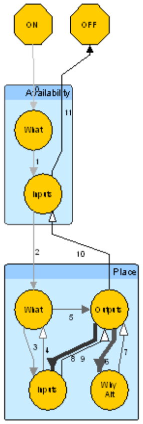
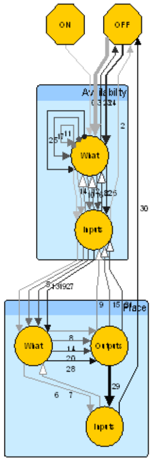
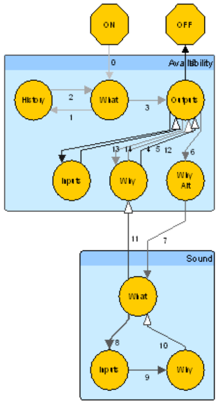
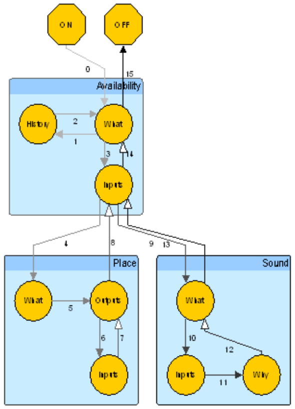
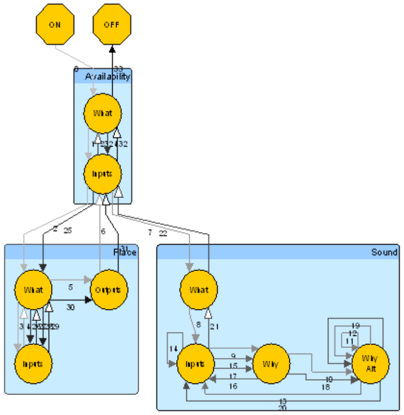
P#	S2	S3	S4-if	S4-situ
07				
08				

P#	S2	S3	S4-if	S4-situ
09	 ON OFF Availability History What Inputs Sound What Outputs Inputs Why Why Alt Definition	 ON OFF Availability History What Outputs Outputs Not Inputs Why Alt Place What Outputs Why Why Alt	 ON OFF Availability History What Outputs Inputs WhatIf WhatIn	
10	 ON OFF Availability History What Inputs Sound What Outputs Inputs Why Why Alt	 ON OFF Availability History What Inputs Place What Outputs Inputs Why Alt Sound What Definition Inputs Why		 ON OFF Availability History What Inputs Place What Inputs Sound What Outputs Inputs Why Definition

P#	S2	S3	S4-if	S4-situ
11	 ON OFF Availability History What Outputs Outputs Nots Inputs Sound What Outputs Why Alt Definition	 ON OFF Availability History What Outputs Outputs Nots Inputs Why Alt Place What Outputs Inputs Why Why Alt Sound What Outputs Inputs Why Why Alt		
12	 ON OFF Availability History What Inputs Sound What Outputs Inputs Why Definition Place What	 ON OFF Availability History What Outputs Inputs Place What Sound What Inputs		 ON OFF Availability What Inputs Definition Place What Outputs Why Alt

P#	S2	S3	S4-if	S4-situ
14				
15				

P#	S2	S3	S4-if	S4-situ
16	 ON OFF Availability History What Inputs Place What Sound What Inputs	 ON OFF Availability What Inputs Place History What Outputs Why Why Alt	 ON OFF Availability What Outputs Outputs Not Whatif Whatif	
17	 ON OFF Availability What History What Outputs Outputs Not Model Whatif Inputs Why Why Alt Whatif Sound What What Outputs Model Inputs Why Why Alt Definition	 ON OFF Availability History What Inputs Place What Outputs Inputs Why Why Alt Definition		 ON OFF Availability History What Inputs Place What Outputs Inputs Why Why Alt

P#	S2	S3	S4-if	S4-situ
18	 ON OFF Availability History What Outputs Inputs Why Sound What Outputs Inputs Why Why Alt Definition	 ON OFF Availability What Inputs Place What Outputs Inputs Why Alt		 ON OFF Availability What Inputs Place What Outputs Inputs
19	 ON OFF Availability History What Outputs Inputs Why Why Alt Sound What Inputs Why	 ON OFF Availability History What Inputs Place What Outputs Inputs Sound What Inputs Why		 ON OFF Availability What Inputs Place What Outputs Inputs Sound What Inputs Why Why Alt