Towards a Semantic Web of Community, Content and Interactions

Anupriya Ankolekar

CMU-HCII-05-103

September 2005

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Katia P. Sycara, Chair
James D. Herbsleb
Robert E. Kraut
Christopher A. Welty, IBM Watson Research Center

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

This work was completed under a grant from the Office of Naval Research, Interoperability of Future Information Systems through Context-and Model-based Adaptation, contract number N00014-02-1-0499.

Keywords: human-computer interaction, artificial intelligence, computer-supported cooperative work, semantic web, open source software communities, semantic web applications

For my parents ...

Abstract

The Web plays a critical role in hosting Web communities, their content and interactions. A prime example is the open source software (OSS) community, whose members, including software developers and users, interact almost exclusively over the Web. The OSS community constantly generates, shares and refines content in the form of software code through active interaction over the Web on code design and bug resolution processes. The knowledge and implementation experiences around the software content are implicit in the interactions in the community discussion forums on the Web. The Semantic Web is an envisaged extension of the current Web, in which content is given a well-defined meaning, through the specification of metadata and ontologies, that can be understood by software agents. This increases the utility of the content and enables information from heterogeneous sources to be integrated. Although the individual components of a Semantic Web are fairly well-understood, there is a research gap in the application of Semantic Web to a specific domain.

This thesis work explores the application of the Semantic Web in the context of a typical OSS community, the OpenACS community, with a focus on the interactions around the bug resolution process. The research answers three questions: How to create a Semantic Web around the OSS community, the software content, and the interactions? How to use the Semantic Web in the bug resolution process? What is the potential impact of the Semantic Web on the bug resolution process, and vice versa? To answer these questions, we developed a prototype Semantic Web system for OSS communities, Dhruv. Dhruv provides an enhanced semantic interface to bug resolution messages and recommends related software objects and artifacts. Dhruv uses an integrated model of the OpenACS community, the software, and the Web interactions, which is semi-automatically populated from the existing artifacts of the community. Comparison of Dhruv's recommendations with historical bug resolution data reveals that Dhruv is able to recommend relevant artifacts for bug resolution messages. A qualitative think-aloud study of Dhruv with OpenACS community members indicates that Dhruv has high potential of being useful to the OpenACS community. Study participants found the enhanced semantic interface particularly compelling.

Acknowledgements

This work could not have been possible without the advice and support of so many people. First and foremost, I would like to thank members of my thesis committee for their constant guidance along the way. I am thoroughly indebted to them for their advice and guidance in turning a vague idea into a full dissertation work. Throughout the ups and downs of the dissertation work, I learnt constantly from my interactions with my committee.

Katia Sycara, my advisor, supported me throughout my doctoral studies, even as I wandered through several research areas, trying to find my own research direction. From Katia, I learnt to think critically about my own research and that of others. I also learnt the importance of setting goals and deadlines as a means to achieving progress on the nebulous road to a dissertation. Jim Herbsleb sparked my interest in the workings of open source software communities. I had several enlightening discussions with him as I scoured the research landscape in this new area. He was truly a mentor to me, especially in the early days of my dissertation research, introducing me to the members of the community and patiently helping me formulate my research direction. I learnt to respect the discipline and rigor of good research from Robert Kraut. He inspired my interest in group work and online communities and intelligent, tailored support for them. From the beginning, Bob maintained a healthy skepticism about the potential of the Semantic Web, which I found very stimulating. Chris Welty provided me with the most direct support in my dissertation work, helping me with several subtle aspects of ontologies. Despite being a remote committee member, he was as involved in the progress of my dissertation as any other member.

In addition to my committee, I would like to Carolyn Rosé for her guidance in the text processing aspects of this work. She gave valuable feedback at crucial stages of this work. I would like to thank the OpenACS/dotLRN community, in particular Carl Robert Blesius, Michael Steigman and Andrew Grumet, for being so supportive of this work and sharing their valuable time when it mattered most. I hope to continue working with this community in the future. I enjoyed being a part of the Retsina Lab and the HCI doctoral students group, whose members have been always been ready for technical discussions and for attending my practice talks at short notice. In particular, I enjoyed working with Naveen Srinivasan, Young-Woo Seo and Joe Giampapa. Massimo Paolucci was always a great colleague, but has now become a good friend.

In the final days of my thesis work, Catherine Copetas, Sharon Burks and Karen Olack stood by in the SCS department to offer friendly help in smoothing the administrative procedures towards graduation and cheer me along. Queenie Kravitz in the HCII made sure that the HCII had appropriate procedures in place for their first doctoral graduate. Marliese Bonk provided excellent administrative support in addition to becoming a good friend. Sumitra Gopal stepped in the last minute to help me find a place to stay in the crucial final days of my dissertation. Preethi Bhat provided me with wonderful hospitality, company and friendship in those final days, for which I will always be thankful. I would also like to thank the Sathe family, who cheerfully hosted me in Washington DC. My family away from home were the Gujarathi family in Boston, who hosted my stay in Boston for the user study and accepted my exclusive focus on my dissertation without question.

I would like to thank my dear friends, Cuihong Li and Kedar Dhamdhere, for their unwavering support. Without their company, my stay in Pittsburgh would not have been as pleasant or as meaningful. Above all, I would like to thank my family. My mother, father and sister were thoroughly involved in my research: cheering me when my research made good progress and bolstering me when I despaired about ever making any progress. Without the emotional support of my family, all the efforts of others would have been for naught.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Online professional communities (OPCs) have flourished in conjunction with the rise of the Web. OPCs are communities of people, who organize themselves and interact primarily through the Web, for work and knowledge sharing. Online professional communities constantly generate, share and refine information through active interaction over the Web as part of various community activities. There are essentially two kinds of OPCs. One type is discussion-oriented communities, which are formed for knowledge sharing, such as Photo.net [Pho]. In such communities, interactions lead to the implicit capture of knowledge in the community Web discussion forums [HS02]. The second type of OPCs are artifact-oriented communities that come together to create artifacts, such as online encyclopædias [Wik] and software programs [MFH00]. In such communities, interactions consist of decentralized, collaborative refinement of the artifacts being created. A major challenge in OPCs is that they generate vast amounts of information as a result of their interactions, but that information is not well-linked on the basis of its content. Thus, community members often find it difficult to coordinate and maintain awareness of other members' activities, leading to wasted labor and reduced productivity.

The research hypothesis is that increased semantic support will be beneficial to online professional communities. With the construction of a semantic model of the content of an online professional community, the interactions of the community and the structure of the community, the activities of the online professional community can be supported.

The Semantic Web [BLHL01] has been proposed and promoted as an enhancement to the current Web. The Semantic Web vision adds a layer of machine-comprehensible *meta-data* over information on the Web and defines *ontologies* that describe the semantics of the meta-data. By thus enabling Web information to be automatically processed based on some representation of its content, Semantic Web technologies can make unstructured or semi-structured Web information meaningful. In the context of OPCs, they facilitate the linking of community information and artifacts on the basis of their content and interpretation. This allows for intelligent support of OPCs in terms of providing better awareness of community interactions and activities.

Thus far, there has been relatively little work in exploring the Semantic Web for OPCs, since the focus of the Semantic Web effort has been on defining basic infrastructure, such as ontology languages and its logical foundations. Interest in Semantic Web applications is now growing, but there are several challenges that must be overcome:

- Generating metadata for existing Web information. Metadata represents the interpretation and distillation of data [Lan88]. However, current metadata tags used on the Web are primarily structural, such as `author` and `date`. More complex data and discussions require metadata tags that are based more on the content and interpretation of information. Creating metadata is not a natural process and requires significant amount of work on the part of the individual or process defining the metadata. However, it is an important process and requires greater attention from the Semantic Web research community. We cannot expect humans to create metadata, so we need intelligent approaches to gather semantics semi-automatically from the data.

- Transitioning an existing community of knowledge and processes into the Semantic Web. It is important to understand how online professional communities can exploit the potential of the Semantic Web and how the Semantic Web can affect community interactions on the web.

The Semantic Web composed of metadata and ontologies primarily enables the appropriate interpretation of information on the Web. In addition, the Semantic Web needs an action-oriented component, that can take appropriate action on the basis of the interpretation provided by the Semantic Web. In the following, we refer to the action-oriented component as an *agent*.

We investigate the research hypothesis in the context of open source software (OSS) communities, which form to develop and maintain software programs and associated artifacts. Prominent examples of software built by OSS communities includes the Linux operating system, the Apache web server, the Mozilla web browser and the Perl scripting language. OSS communities are among the most complex OPCs, because their core task of software development is an inherently complex activity requiring a substantial amount of coordination. OSS communities share characteristics of both discussion-oriented and artifact-oriented communities. By selecting OSS communities for consideration, we can investigate semantic support for both types of OPCs.

Within OSS communities, we focus on the bug resolution[1] activity. Bug resolution is one of the most difficult software development activities and yet crucial to the quality of the software produced. Community members typically need to thoroughly understand the software source code and to be aware of the activities of other community members in order to resolve a bug. Furthermore, due to its concrete nature, bug resolution is more suitable as a research context for development and evaluation purposes than other community activities.

We explore the application of the Semantic Web in the context of a typical OSS community, the OpenACS community, with a focus on the interactions around the bug resolution. We seek to answer three research questions:

1. **How to create the Semantic Web for an online professional community?**

   How can we create a Semantic Web around an OSS community, the software content of the community, and their interactions? To answer this question, we create

[1]Within this document, *bug resolution* refers to the process that the OSS community goes through in order to identify, understand and fix a bug. It includes additional steps that may be taken to avoid the occurrence of similar bugs in the future. In contrast, *bug fixing* denotes the process that a single developer goes through in order to develop a solution for a recognized bug.

a prototype community semantic web for the OpenACS OSS community, called Dhruv[2]. To build the prototype, we define ontologies for the content of the community, its interactions and its members. We process the community information to generate metadata semi-automatically and use several heuristics to create semantic links across community objects and artifacts.

2. **How to use the Semantic Web in the context of an online professional community?**

   How can we use the Semantic Web in the bug resolution process? To answer this question, we develop an modified interface to the community bug tracking system that exposes the Semantic Web information and processing to bug resolution participants.

3. **What is the potential impact of the Semantic Web on the online professional community?**

   What would be the impact of the Semantic Web on the bug resolution process, and vice versa? In order to answer this question, we ask members of the OpenACS community to evaluate and assess the enhanced bug resolution interface presented by Dhruv. In addition, we evaluate its recommendations of related artifacts by comparing them with the artifacts the community actually used in historical bug resolution instances. Within this, we explore how the Semantic Web can evolve through people's participation in the bug resolution process.

The contributions of this work are to establish the relevance and viability of the Semantic Web to the OSS community and a framework for creating a Semantic Web for a specific OSS community, through the development of a model and an associated Semantic Web knowledge base of a typical OSS community, their software content and interaction processes. In addition, we demonstrate that a hybrid approach combining various information sources and techniques for judging similarity between artifacts leads to better

[2]In Indian mythology, Dhruv is the north star or pole star, the only stationary point in the sky, around which the entire universe is said to revolve

recommendations by Dhruv compared to ‘purer approaches. We also present four ontologies for OSS communities and techniques to semi-automatically generate Semantic Web metadata and heuristics to semantically link various community artifacts and objects.

1.1 Organization

In the next chapter, Chapter 2, we discuss the motivation for constructing a Semantic Web for OSS communities in detail. In particular, we describe online professional communities, especially OSS communities, in detail: the content they generate, the interactions and activities within the community and the roles that various community members take on. OSS communities face challenges in maintaining accurate and comprehensive documentation, coordinating and maintaining awareness of community activities and in identifying expertise. We present a scenario of the possibilities presented by a Semantic Web for OSS communities and discuss how the Semantic Web could address the challenges faced by OSS communities. Finally, we discuss the individual components of the Semantic Web required to realize a community Semantic Web. We also examine the related work in this area and how the Dhruv differs from them.

Having motivated a Semantic Web for OSS communities, we answer the first research question in Chapter 3. The chapter describes the creation of the community Semantic Web prototype, Dhruv, and its components, namely various kinds of ontologies, techniques for semi-automated metadata creation, heuristics to link community objects and artifacts and heuristics to generate message recommendations from cross-linked objects and artifacts.

In this chapter, we also describe how the various information sources and heuristics are combined in Dhruv to present an enhanced bug report interface for the community. Thus, this chapter also answers the second research question of how to use the Semantic Web in the context of an online professional community.

The final question is answered in Chapter 4, where we present the evaluation of the Dhruv prototype. We use two different ways to evaluate Dhruv: a comparison of Dhruv’s recommendations with historical data and a qualitative user study to evaluate Dhruv’s information interface.

In the last chapter, Chapter 5, we discuss the results of the evaluation presented in the previous chapter and interpret its implications, for the Semantic Web and for the support of OSS communities. We end with a list of the contributions of this work.

Chapter 2

The Semantic Web for Open Source Software Communities

In this chapter, we discuss the motivation for constructing a Semantic Web for OSS communities in detail. In Section 2.1, we describe online professional communities, especially OSS communities, in detail. Within this section, we discuss the nature of OSS communities and their members in section 2.1.2, their interactions and activities in section 2.1.3 and the various kinds of content they generate in section 2.1.4. Examination of the activities and nature of OSS communities leads us to note several challenges that they face (section 2.1.5), namely (a) maintaining accurate and comprehensive documentation, (b) coordinating and maintaining awareness of community activities and finally (c) identifying expertise. In the following Section 2.2, we illustrate these challenges by means of an actual bug report in a chosen OSS community. We then describe a scenario of the kind of semantic support that could enhance community processes, leading to more effective and efficient performance of OSS community activities. In the final section, Section 2.3, we discuss the individual components of the Semantic Web required to realize a community Semantic Web, namely metadata (section 2.3.1) and ontologies (section 2.3.2). We end this chapter with a discussion of related work in Semantic Web applications (section 2.3.3).

2.1 Online Professional Communities

The Web is increasingly becoming a substrate for the formation and sustenance of online professional communities. There are a wide variety of online professional communities: ranging from photography to to hardware and software support groups [Pla, Ubu, Moz]. Professionals come together online to share information, experiences and knowledge and provide technical support to each other on the Web. The professionals are typically highly qualified in a field and they get together to participate in technical discussions or to produce technical artifacts. One very successful example of a community which collaborates for artifact creation is the Wikipedia [Wik] project, a collaboratively constructed online encyclopædia. The quality and scope of its entry attest to the commitment and professionalism of its contributors. Online professional communities are thus distinctly different from recreational or social communities, although they may at times fulfill similar functions for certain individual members.

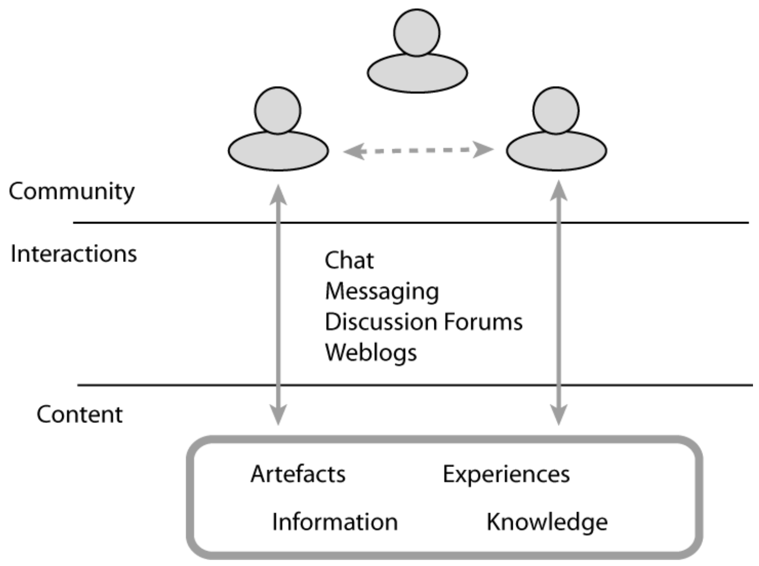


Figure 2.1: Online professional communities

How do online professional communities collaborate to so successfully? The anatomy of an online professional community can be viewed as depicted in Figure 2.1. As people interact with each other in the community, their contributed information, knowledge and experiences get stored in the online community archive. Similarly, collaboratively built artifacts and the knowledge and experiences of the construction process also become a part of the community's content. The archived content thus becomes a valuable resource for current and future members of the community, a kind of collective good [Mil00]. As the archive grows through community interactions, it For discussion-oriented communities, the archive is in fact the primary product of the community. becomes increasingly valuable for the community.

The community interactions that cause the growth of the community archive typically take place through a variety of Web media. The most ubiquitous medium is probably the mailing list, typically archived on the Web. A mailing list is essentially a set of email addresses that can be collectively sent email. Mailing lists are sometimes the only means of interaction within an online professional community. This is particularly true for discussion-based communities. Communities with a website will often also have Web-based discussion forums. In this case, people use a Web browser to navigate to the forums and post messages. Most Web discussion forums consist of multiple topic forums, each with multiple threads. Thus, discussion forums display threaded discussions and allow people to start a new thread. Recently, chatrooms are becoming increasingly popular as a means of brief, synchronous interaction, particularly for OSS communities. Some communities may not use any of these tools, but they all still do provide some means of interaction. For example, Wikipedia has none of the above discussed interaction media. However, each page in Wikipedia has a history of past modifications made to the page by the community and short messages explaining the rationale for the modifications. This is essentially an interaction medium for the community and controversial pages often have 'heated histories'.

The most prominent example of online professional communities, however, is probably open source software (OSS) communities, which form around the source code of a software program. Despite the complex and interdependent nature of software development [KS95], OSS communities have been remarkably successful, with several OSS software projects comparing very favorably with commercial offerings. Some of the most prominent OSS communities are the Linux operating system [Lin], the Apache software collection [Apa] and recently, the Firefox web browser [Fir]. OSS communities display characteristics of both discussion-oriented and artifact-creating communities, because the communities engage in a complex creation activity which requires a fair amount of coordination through discussion. In the next section, we examine OSS communities and the nature and means of their collaboration in detail.

2.1.1 Open Source Software Communities

OSS communities form around an open source software program, a software program whose source code is publicly available. The communities revolve around the program code, working to develop the program further, to fix bugs (defects in the software), to provide support on using the program, discuss future evolution of the software and so on. Essentially, the community performs the whole gamut of software development activities. The software they work on could be almost anything, from computer operating systems [Lin] to graphics libraries [XFr] to editors [XEm] to scientific software [R]. The code could be something written by one of the current or past community members [Lin, Moz] or a commercial program passed into the open source domain [Ope].

OSS communities are virtual communities and accordingly, interact primarily through the Web. They use a variety of communication tools, including mailing lists for technical discussions and support, a bug tracking system for monitoring and fixing bugs, a CVS code repository for storing a common version of the source code. In addition, communities often use chat for more real-time communication. Archives of all past activity is usually available through the community website and can be browsed. Thus, the OSS communities use the Internet primarily as a communication and storage medium.

The OSS community brings together a loose collection of volunteers: the active developers of the software, the end-users of the software, and anyone who has an interest in the software. They organize themselves as a community, creating roles for themselves and performing administrative functions, such as voting, marketing etc., in addition to the main software development activities, such as source code design, code implementation, program maintenance etc. Open source software communities face the challenge of maintaining awareness of other developers so that they can coordinate their own work with others. Coordination becomes particularly important when taking a long term view of developer activities. Activity-centric view is crucial to providing adequate support, particularly since OSS communities have severely restricted communication compared to co-located software development: they communicate primarily through artifacts and email discussion.

Thus, like online professional communities, OSS communities can be viewed as having three layers (see Figure 2.2): a content layer of software code, bug reports, documentation etc.; an interactions layer that builds on the content, as people interact through bug tracking systems and web forums to participate in activities around the code, such as software development and bug resolution; and finally a community layer that is formed through the interactions that take place around the content. The community layer contains people and their various, dynamic roles in the OSS community.

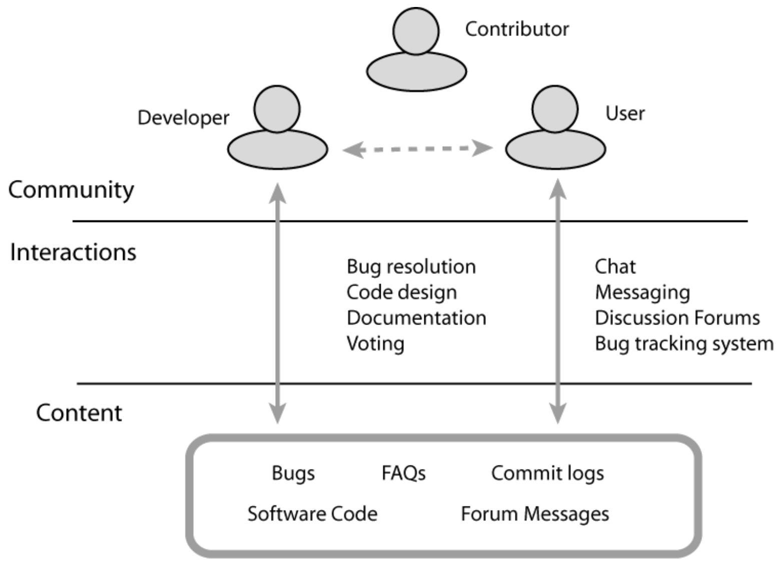


Figure 2.2: Community, Content and Interactions in an OSS Community

2.1.2 The Community

Open source software development refers to a kind of geographically-distributed software development, where strangers from remote corners of the globe collaborate on building software through the Internet. Who are these people? What brings them together? And how do they manage to work together without knowing each other?

The members of an open source software community are diverse, comprising professionals, hobbyists, consultants, students and even researchers themselves. They take on various roles within the community, such as that of users, developers and contributors. Users are primarily consumers of the output of the open source project. They use the software and seek the help of more experienced members of the community, when they face difficulties or have technical problems in the use or installation of the software or encounter bugs (defects in the software). Users often seek help by posting a question to a discussion forum or asking developers directly in the community chatroom. As the problem is resolved with the help of others in the community, their interaction gets archived and is available to other users facing the same problem later on.

The help providers are typically so-called contributors or developers. Developers are usually both consumers and producers of the output. Core developers [vKSL03] are deeply committed to the project and the community and will often spend significant amounts of time providing help to users, enhancing the source code, guiding the evolution of the software by reviewing contributed code and encouraging new contributing members of the community. Core developers also have the responsibility of selecting a subset of code contributions to be ‘committed’[1] to an ’official’ release of the software.

The people to contribute code to the community are typically technically-oriented users of the software, who are also interested in the general development of the project. They are likely to download the most recent (possibly unstable) versions of the software, actively report bugs, and submit code, either to fix bugs or to provide further enhancements to the software or to contribute patches. As contributors get more involved in the com-

[1]The software source code is typically stored in a shared repository. Saving code to the common repository is also known as ‘committing the code’. Access to the repository is limited to the core developers.

munity, they make take on and receive greater responsibilities, such as commit privileges and greater say in shaping the source code produced by the community. Most core developers of the community were newcomers to the community [vKSL03] at one point. Through their commitment and contribution, achieved recognition and respect from other community members. As a member of OpenACS explains:

> *<B> there are alot of people _using_ openacs to get work done, and when they can they contribute stuff back.*

However, there are often no binding ties between people and the community. Having achieved their goals or found greener pastures, community members may gradually drift away from the community. This is evidenced by the relatively Most open source developers spend only a couple of years in any given community before moving to other communities. In addition, most open source developers participate in multiple projects simultaneously [Gho03].

In general, in OSS communities, the software user community is routinely involved in critiquing and reviewing the source code of the project. By doing so, they also provide strong input on code features and implementation design [DGMN02]. It has been suggested that the primary motivation for people involved in the process is a quest for knowledge and peer recognition. Recent surveys of open source software developers, however, suggest that the motivations of individual community members in open source software development are numerous [Gho03]. Community members may be motivated by educational goals, such as learning a programming language or gaining experience in building certain kinds of systems. They may also be ideologically committed to the open source software movement and desire to participate in an OSS project. Recently, business motivations are also coming into the picture, as developers are sometimes paid by large corporations to participate in OSS projects [IBM]. As a core developer in an OSS community [Ope] states:

> *<C> there's a lot of code here, the categories package is relatively new, and most of us learn packages as we need them for our client work*

Many complex OSS projects have spawned consulting companies that customize the OSS software developed in the community to provide solutions for businesses.

<K> ... I really think that you might find that Claudio has already done some or all of what you need. They have written the guts of a complete ERP system and he has just uploaded the code of a version that he has rolled out to clients. That means it has probably been largely bug...
<K> ...fixed too :-)

Given that it is in the companies' interest to have the OSS community flourish since they can then use the community-developed code, such consulting companies often have a few employees who are prolific contributors to the OSS community.

Given the wildly varying motivations of OSS community members, their involvement with the project also varies widely. Each member of the community brings his own experiences, knowledge and practices to bear on the project, which is molded and shaped almost directly from the myriad influences of individual members. Since OSS communities rely almost exclusively on community members to promote, participate within and drive the development of open source software [AB02], the collective input of community members shapes the software, the community and its work processes. Thus, OSS development has evolved its own set of Web-based tools and practices [AB02] to suit their unique environment and each community customizes these to fit their own needs and culture. Thus, while actors directly shape the output and the processes within the project, they are themselves influenced by the project environment, existing processes and tools [HS02].

Guided by the changing requirements of the project and their own dynamic preferences, the contributors and core developers of a community naturally assume various working roles. The roles that people take on depends mainly on their individual motivations, which drives their perception of the project and their view of the trajectory of the project. There are little or no compulsion from the community itself on any individual member to perform any particular work. The roles that people perform include bug fixing, code development, bug triage, code design and writing documentation and are closely tied to the activities of the community. The particulars of the role depend on people's understanding of certain portions of code, and the various processes that determine how the project functions. Community members also progress over time in assuming roles with greater responsibility as they show long-term commitment and capability in the project. Thus, community members may start out with bug reporting, then progress to submitting patches and participating in technical discussions to ultimately becoming a core developer in the community.

2.1.3 The Interactions

Open source software development typically consists of a series of activities around the software code [MFH02]. These include the discovery that a bug exists or that new functionality is needed, determining who among the pool of active developers will work on the issue, identifying a solution to the issue, developing and testing the solution, (if needed) presenting the code changes to the core committers for review, and committing the code and documentation to the repository. In fact, it has been suggested that most open source software projects operate in what would conventionally be regarded as the software maintenance and evolution phases of the software lifecycle [HS02]. The original code that seeds an OSS community is typically developed by individuals or by commercial software development teams and then contributed to the open source domain. It is rare for an OSS community to form without any seed code.

In addition to these software development activities, there are several administrative and support activities associated with maintaining the OSS community. With the OSS community, there is a strong culture of ‘making it public’, i.e. conducting all community interactions, such as answering questions, discussing plans and design details and reporting on project status, in public locations, such as the community mailing lists and other discussion forums [GPS04]. Most such explicit interactions in an OSS community are discussions about some bug or a design detail or the best way to implement a new feature and any person can freely participate in such discussions.

In order to participate effectively in the above activities, OSS community members need to understand who is working on what in the community and how their work affects other community members. This is also known as *group awareness* [DB92]. Such knowledge allows people to coordinate work effectively, anticipate other members' actions, discuss tasks and locate help [GG02]. There are three primary ways that an OSS community maintains such group awareness. These are [GPS04]: reading developer mailing lists, reading real-time chat and watching commits from the code repository. In the following, we discuss the interactions of the OSS community as they take place around various interaction tools, namely the code repository, the bug and issues tracking system, the discussion forums and chatrooms.

Code

At the heart of every open source software project is its code, maintained in a central repository, usually with the help of a version control system, such as CVS [Cvs]. The central repository has been conceptualized as a "walled server" [HS02], where the "wall" is the set of open source collaboration tools and practices that allow anyone to browse through the code, however, severely restricting the contribution of code to a few core developers. All the code contributed must pass through one of the core developers with commit privileges. Once commit privilege has been granted to a developer, though, there is no restriction on where they can contribute. Developers with commit privileges can work on any area of the code they wish to. As a developer in Apache put it: "all committers are responsible for all parts of the code" [GPS04]. As another developer put it [GPS04]:

> *"Responsibility is a strange concept in a collaborative volunteer project. With most things there are several people who know their stuff, so there's no clear concept of responsibility. The exception is of course where someone's name is down against something. For example, I put my name against the <xyz> package as its maintainer, and so I'm responsible for it. When I commit my new port, I will be responsible for that."*

The source code therefore provides a setting for community interactions, as various developers may modify the same code within a (short) time period. Through these interactions, the developers are constantly negotiating a shared understanding of the software code they are developing. The constant interactions between developers result in a continuous redesign of code structures, the code architecture and coordination processes.

Bug resolution

Bug resolution is a very important activity for OSS communities since it determines the quality of the code produce. It is often also a precursor to an official release of the source code. As a developer remarks in a message about the official release process of OpenACS[2]:

> *Many [bug fixes] came from volunteers and by professional OpenACS coders in their "spare time." The single biggest delay in getting 5.0 shipped has been getting bugs fixed.*

Bugs and feature requests are usually tracked by means of an issue tracking system such as Bugzilla [Bugb] or the OpenACS Bugtracker [Ope]. Bug fixes or patches are sometimes also submitted through the bug tracking system. The community is encouraged to use the bug tracking systems, since it then becomes a single access-point for all modifications to be made to the code.

> *<H> I don't have cvs access, but I will mail a patch off to the maintainer/author.*
> . .
> *<B> put it in bugtracker!*

In most OSS bug tracking tools, each such bug becomes a message board centered on the issue [HS02]. However, bug fixing is not as exciting as code design and development. Consequently, it often tends to get neglected. This criticism of a community by a member is true to varying extents for all OSS communities:

> *there is a tendency to constantly improve by redoing, and very little improvement of what is already in existence. as a result, old bugs sit and get older, new bugs are introduced, and the featureset is a hodgepodge of unfinished trinkets.*

[2]Notes on the .LRN release process (from Joel) 01/28/04 09:18 AM

Part of the problem is that since the community has collective responsibility for fixing bugs, no person is explicitly assigned to any bug. Every now and then, for example, before a software release, people will go through the bugs posted on the bug tracking tool, looking for bugs they can fix. This places the burden of finding bugs that can be fixed squarely on the people who can fix them. Thus, there can be a long delay before the bug gets to the attention of developers who can fix the bug. Of course, if a community member is really keen to get a bug fixed, they can always specially request help with the bug on the discussion forums. Community members are generally quite responsive and try to help out as far as possible.

The bugs that do get submitted to the bug tracking tool may themselves not be genuine bugs. A common example of an invalid bug is one that cannot be reproduced. Determining that such bugs are invalid wastes developer time and clutters the bug tracking system. An OpenACS developer explains the procedure for dealing with such bugs in the community chat room:

> *<J> people who are working on bugs should close them or at least comment that they think they have fixed them... there's all these open bugs that actually seem to test OK; so far I haven't reproduced any bug*
> *<A> J: i hope you're marking the bugs as "unable to reproduce" or "more info" ... and explaining that you can't reproduce it/them ...*
> *<A> bump it down from open to whatever state tells the person who reported the bug in the first place that it's not going to get acted upon without further info*
> *<A> that may encourage them to update their copy and check for the error again - and if it's really fixed, they can mark the bug as fixed or comment on it appropriately as well.*
> *<A> then other developers won't waste time looking at the same bug and reading it top to bottom only to get to your comment which says it's not reproduceable.*

Receiving duplicate or invalid bug reports is unfortunately quite common. In the Apache project, a few dedicated developers would usually go through the bug reports, mark duplicate bugs, remove mistaken bugs, fix simple bugs quickly, review and commit patches, and forwarding reports to the developer mailing lists if the bugs are considered critical [MFH02]. However, not all OSS projects can afford dedicated developers to triage the bug reports and sorting through the bug reports remains a major problem. As an OpenACS developer explains:

> *[Triage] entails looking at all incoming bug reports and adjusting priority, severity (if the reporter got it wrong), and fix by. This bounced around a lot - the OCT nominally has a rotating monthly triage duty, but that doesn't seem to have been super-effective. Mostly random people went in (to the OpenACS.org bug tracker) and triaged from time to time ...*

The responsibility for fixing specific bugs can sometimes bounce between several developers or groups of developers before eventually being accepted. Sometimes developers will themselves bring bug reports to the attention of people, who can fix the bugs. Once the bug has been fixed or the enhancement developed, the bug tracking tool is searched for similar reports, so that those can also be closed.

Discussion forums and Chat

Although the code and bug reports do offer venues for interaction, by far the most prolific one are the mailing lists or discussion forums of the project, which are rife with technical discussions. The discussion forums represent the pulse of the open source software project, where developers working on different areas of the project report on their work, seek and provide advice and get updated about developments associated with the project. These are the place to seek help on a wide variety of things: help with installation problems to discussions on bugs and design details to proposed extensions to the code and the best way to implement them.

Since open source software is built and maintained by large groups of people, it is important for developers to understand what others in the group are doing and what they know to work successfully in groups [TSL93]. Due to the public nature of community discussions, all developers implicitly becomes peripheral participants in all discussions [GPS04]. By ‘overhearing’ other conversations and by seeing who is talking about what, people know of each other’s expertise and history of interactions [Smi02]. This lets them know who the right person to talk to is and help route other queries for help:

> *<B> sorry I don’t have the categories package memorized yet.*
> *<C> jeff or timo could answer your questions better*

As a peripheral passive participant in discussions, people also keep track of the main issues and what was discussed by whom:

> *<D> okay, B will know. There’s a link somewhere... I’ve seen it discussed here*

Mailing lists and chat also allow people to directly reach the experts in an area, simply by initiating discussion. Since messages are sent to the whole group, the right people identify themselves by responding and participating in the discussion [GPS04].

By far the most important part of what happens on discussion lists is question answering. Although everyone has access to the source code, the code does not reflect the unwritten design choices and principles underlying the implementation of the software. Since an OSS community always consists of a large proportion of new comers, who are still learning the ropes, veterans in the community then act as mentors and explain such design detail information in great detail. Like mentors in the industry [Ber93], the veterans will often answer much more than the initial question. They will emphasize the purpose of a function, how it interacts with other procedures in the system, rules such as when and what values are expected by certain parameters. Such information is hard, if not impossible, to find purely by studying the source code. In addition, experienced community members will provide rationale for implementation decisions, compare related software objects and expound on performance issues, exception behavior etc. Thus, discussions are a vital storehouse of knowledge and experiences.

Discussion forums and chat rooms are somewhat similarly used. In addition to technical discussions, both these channels are used to discuss whether a problem is a bug or not.

<A> B: should i file a bug?
<B> sure!

So, discussion in forums can precede or be contemporaneous with activity and discussion in the bug tracking system. Discussion forums and chat are also used to ask and learn about procedures and standard operating practices of the community:

<B> actually to submit a new package you should post on the forums about it.
<B> that will let you know if anyone is interested

This also allows community members to be aware of the activities of individual members, thus helping avoid redundant work. However, there is ultimately no infallible way to avoid duplicate work. The community relies on the vigilance of individual members to make sure that no one else has already done the work they are currently doing. Unlike discussion forums, chat rooms are also used for real-time coordination between developers. Therefore, such interactions are common:

<E> hi B
<B> hey
<B> where am I supposed to be looking at your code?

<A> B: hey... i have a question for ya.

<C> actually I was hoping to see B here

Chat rooms are also used for real-time code discussion and development, although it is admittedly harder to discuss code implementation over chat than in person.

<B> what would the code look like?
<B> this be alot easier if we were in the same room.
<B> but let's see what we can do.

2.1.4 The Content

The many kinds of OSS community interactions generate a huge amount of explicitly captured content, in the form of discussions, documentation, bug resolution activities etc. in addition to the code itself. The community archive of content is the core nucleus around which the entire community revolves. The community forms around the content, draws from the content and contributes back to the content.

Despite the numerous kinds of content produced as a result of OSS community interactions, the most salient artifact and the most authoritative representation of the state of the project is the software source code [AB02], which is stored in a common repository, usually through 'Concurrent Versions System' (CVS) [Cvs]. However, there are several challenges to the smooth extension and refinement of the source code by the community:

1. *Understanding the source code* plays a large part in fixing bugs and producing enhancements to the software. The open source developer must understand the software and its relationship to the domain, requirements, design, end users, documentation, comments, other maintainers, future changes, etc. In fact, most developers in conventional software development spend as much as half their time in meetings (understanding what each other are doing and what things need to be done) for exactly this reason. Furthermore, during the time they spend doing maintenance, 60% or more is spent searching through the source code for the information they require to understand and complete their task [Sel90]. As a result, maintainers typically rely on the documentation and comments in the source code to guide them in understanding the software [Wel95, AB02].

2. In addition to the source code, many OSS communities do have some entry-level documentation for new community members. However, such *explicit documenta-*

tion is rarely very comprehensive due to the high level of effort required by OSS community members to prepare this with no direct benefit to the documentation authors themselves. Good documentation does benefit a community by making it easier for new community members to use and contribute to the source code. Its importance in attracting new community members is well-recognized by the developers themselves:

> *<A> if you write docs, you'll make the toolkit friendly, you'll get users, which will give you more contributions (in terms of cash and code)... which in turn will get more people, some of which will be willing to document their contributed code or fixes.*

3. Despite such recognition, *writing documentation is usually low on the priority list* of core community members, because of the pressure to produce new code and release it. On the other hand, most OSS developers are very conscious of value of source code comments as a means of communicating the intent of the code to other developers. The source code comments are therefore often the best form of documentation available, barring discussion messages discussed below.

4. Since open source software projects typically experience high turnover [Gho03], the *code is often designed, implemented and maintained by different groups of people.* Thus, each new developer needs to understand not only what the original designers and implementors of the software did, but also the changes made by previous developers as part of the maintenance of the system [Wel95]. The new developer then re-interprets the requirements for the software and adds his own modifications. The layering of such modifications can lead to contradictions, to bugs even in code that was problem-free before.

5. Perhaps even more so than in traditional software development, *the code repository commit logs become absolutely vital to keeping track of who is modifying the source code and where in the source code the modification is taking place.* Many developers subscribe to commit log notifications. Whenever any change is made to the code base, these developers are sent an email with details of the change. The CVS system

allows developers committing changes to associate a message explaining the nature and reason for the modification. These commit log comments are always used by OSS community members and are an important way of reporting their actions to the community at large. In fact, Gutwin et al. [GPS04] found that the five primary sources of awareness information in an OSS community were: package maintainers, code repository logs, issues and bug trackers, asking other developers and project documentation.

Given that discussions play such a major role in the community (see Section 2.1.3), the messages in mailing lists and discussion forums often perform an implicit role of providing documentation. The documentation is more of a semi-structured 'stream of consciousness' [HS02], in the sense that the discussions are situated in numerous threads within topic- or issue-specific mailing lists. Community members often summarize some of the discussion content into documentation and FAQs on an ongoing basis.

As we have seen, community activities are distributed over several tools and artifacts, which are not closely linked. People do refer to artifacts in their discussions, by inserting deictic references [DB03] in the messages they write. Since the entire community archive is accessible through a Web browser, these references are often HTML links. The links could refer to archived discussion threads, bug reports, CVS commit logs, source code files or documentation. In fact, because the community archive is Web browsable, each of these artifacts cab potentially have embedded HTML links to each other. Community members do at times make use of this facility to insert links manually. Sometimes, the tools themselves are constructed so as to support this linking process. For example, a bug tracking system used by the OpenACS community, Bugtracker [Buga], has a patch tracker, which, upon creation of the patch itself, prompts for the bug report the patch is fixing. If a developer mentions a bug report, then the bug report and patch are automatically linked and presented together.

2.1.5 Challenges

We discussed how the community layer is very fluid in section 2.1.2. People participate within the community based on their own personal needs and requirements, which makes the interactions of the community very spontaneous and unpredictable. As a result, the content layer in Figure 2.2 is very dynamic, yet usually somewhat haphazardly developed. Some areas of the code may receive significantly more attention than others, depending on the interest of the community as a while.

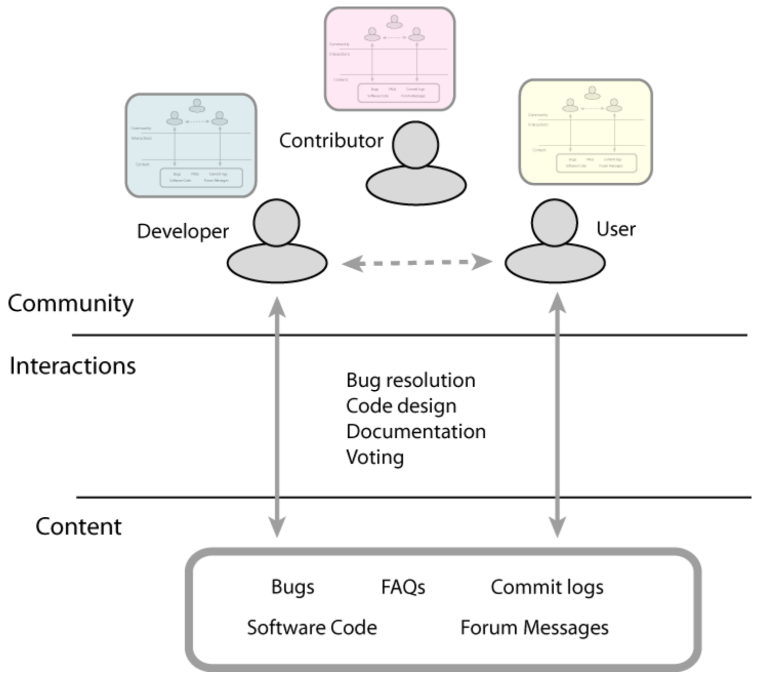


Figure 2.3: Each community member has an own understanding of the semantics of the three layers of the community

Within such a complex, dynamic environment, people participate in the community based on their understanding of the semantics of the community (who talks to whom, who does what, who knows about what), the content (which bugs fix which code, how does the code structure fit together, why was this code implemented this way) and the interactions (who said what, who did what). Through long-term participation in the community, people gain an ever-finer understanding of the three OSS community layers. This understanding of the community, content and interactions, as shown in Figure 2.3, is therefore different for each community member and modulates their interactions with the rest of the community. This has two implications. First, this semantics may not be available to those who have had little interaction in the community, for instance, new community members. Secondly, the semantics is not shared. It is fragmented into the minds of community members.

Most of the semantics is not reflected directly by information in the three layers. It is contained solely in the minds of community members. If people's semantics of the information in the three layers could be expressed explicitly and associated with the information, it would make the information much more meaningful (see Figure 2.4).

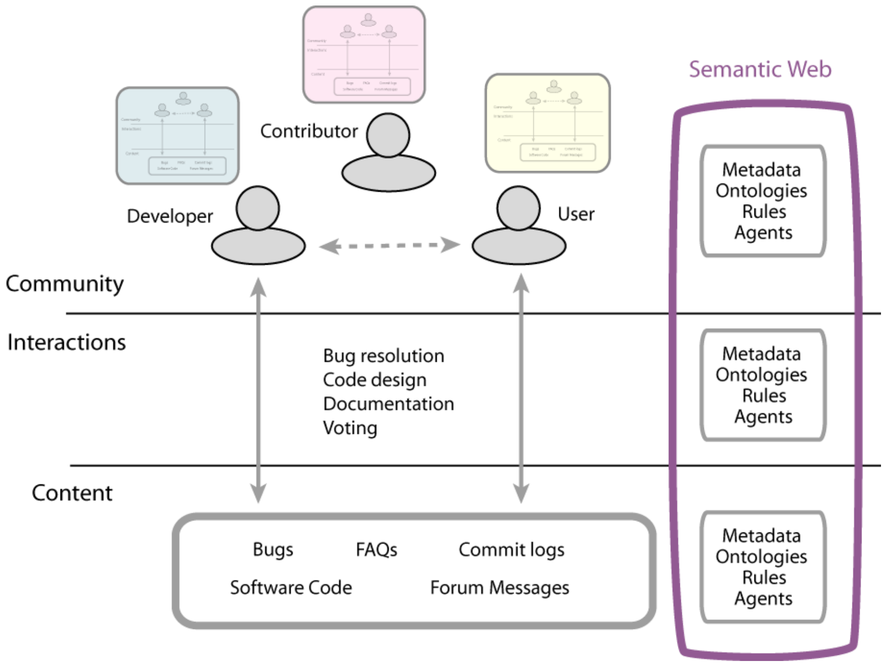


Figure 2.4: Making people's semantics explicit

Making community semantics more explicit would enable new community members to make use of it without having to acquire it gradually. This would make their interactions with the community more meaningful and less wasteful. Furthermore, the explicit semantics would lend itself to machine-processing. Making people's semantics accessible to machines allows them to process the community archive, mine it and present information to the community in a much more intelligent and contextually-appropriate manner than is currently taking place. In other words, the OSS community's web of information can become a *semantic web* of information.

In the following subsections, we discuss several challenges for OSS communities and how the semantic web can ameliorate each of these challenges.

Documentation

The lack of adequate documentation is a long-term problem hounding many OSS communities. Many OSS community members are interested in a particular OSS software and want to contribute to it. However, new developers find it difficult to understand the semantics of the code. For example, they are stumped by the lack of clear tutorials or documentation appropriate for beginners. As this new developer in OpenACS laments:

> *<A> i wish i knew what a category subtree was....*
> *<A> i'm reasonably sure i want to use them*
> *<A> but just can't be sure whether i do or don't, and worse yet, if i did, how to.*

By persistently browsing the code or through trial and error, some OSS developers do manage to learn how to use the software. The same developer again:

> *<A> and you guys sure do make the coolest part of the damn toolkit impossible to find*
> *<A> i just discovered* `util_user_message`
> *<A> wow. it's like brilliant.*
> *<A> it's like, completely undocumented.*
> *<B> A: oh no, you disccovered our secret*
> *<B> it is documented*

<*B*> *in* `ad_returnredirect`
<*B*> @ `http://mark.stosberg.com/Tech/darcs/cvs_switch/`

In this case, they may find what they were looking for is partially documented somewhere, but was just not accessible when they needed it. The documentation is fragmented and there is no link from where they were looking to where the documentation actually exists. As developer A continues:

<A> B: ugh, yeah - documented, as a comment on a pseudo-random manual page.

Even if code does have documentation, it may not be satisfactory for new developers. Code documentation is only written through people volunteering to write documentation. This happens much less than would be required for full-fledged documentation. As developer B quips:

<B> you'll notice there aren't a legion of volunteers fixing the documentation.
...
<B> got any brilliant ideas how to get people to write documentation?

Getting people to write documentation is difficult and may not even be required. Certainly, the current situation in OSS communities could be ameliorated by accessing the documentation implicitly present in discussion forums and chat logs. As discussed previously, a significant portion of the documentation is anyway collated by community members from the community discussions. Instead of waiting for and relying on people to do so, the semantic web can go right to the source of the documentation, the discussions, and collate them automatically. As developer A exclaims:

<A> as painful as it may be - i think it's worth dumping this notion of the code is the comment...

Instead, the notion of 'the discussion is the documentation' may be the right way to go. Due to the dynamic and prolific nature of discussion forums, many things get discussed repeatedly with new information added every time. If discussions about the same topic were linked together, a developer seeking information on a topic could browse all the discussions on that topic. At present, the discussions are not easy to search through. As the value of archived community conversation increases [Mil00], so does the need for better ways to browse and search the contents.

Although there are community members, who are reasonably satisfied with current procedure of documentation:

> *<H> with good use of grep i find that i rarely need real docs. . .but a kind of overview of a package wouldn't be a bad idea.*

most programmers are unhappy about it. If OSS developers still rely on the documentation and comments in the source code to guide them in understanding the software, as happens in traditional software development [Wel95, AB02], it may be because the discussions are not adequately performing role of providing documentation. As developer A continues about comments in the source code:

> *<A> it's nice for generating man pages for individual procs, but it fools people into thinking they are documenting their package when they are just writing docs for the API.*
> *<B> I think we have mountains of [in-code] documentation that doesn't help*
> *<B> and are missing all the stuff we need.*
> ...
> *<G> api docs are okay, but boy does it need more detail*

Coordination and Awareness of Community Activities

Most OSS activities involve interactions about multiple artifacts and take place via a number of interaction tools [GPS04]. One reason for this is that various components of OSS activities involve examining different kinds of information, such as bug report data or discussion messages. Most OSS tools, like most desktop applications, present a data-centric view rather than an activity-centric view. People in an OSS community thus commonly need to refer each other to information in a different tool, as the following snippet of OpenACS chat room logs illustrates:

<B> (also what did you think of the stuff I posted about forums, I think it does pretty much what you want already)
<M> I didn't know that you posted
<M> have to read that
<B> oh :)
<B> yeah do that. it should be in the log.

Different tools also target different audiences. For example, in the OpenACS community, a majority of community members browse and post in the discussion forums. A much smaller proportion of community members participate in the chat rooms, as the following snippets of chat room logs reveal:

<B> I'd rather you ask on the forums
<B> since you'd get a better response apparently all the smart people are taking the day off of IRC :)
...
<B> I think you have enough information to post to the forums, and get feedback

The main reason why people need to constantly switch between tools is that the tools and the artifacts contained within are not linked together semantically. In every different tool, people need to browse through or search for information relating to the same topic all over again. In essence, community information is organized chronologically and disjointly in the archive of each tool. In terms of the community semantic web, people need to go back and browse the history to find important or relevant information themselves, because the semantics of the information is only comprehensible to people. More often than not, people don't go back.

Much of the information related to a topic is therefore lost in the depths of the archives if it was authored sufficiently before the present moment. Consequently, although the community archive as a whole is a huge repository of content, communities have difficulty in deriving maximum benefit from the archive. For example, there are indications that the current search functionality and documentation are insufficient to cater to developer needs:

<Z> and openacs.org certainly needs a better search
<M> and faster
<B> M: its slow because noone uses it :)
...
<A> uhmm.. is there any way to search in bug tracker?
<G> bug tracker needs a few features i think :)
<A> does it REALLY not have search?

The current situation could be considerably ameliorated by adopting an activity-centric view of the community interactions instead of the current data-centric view. By linking semantically related information across artifacts, a community semantic web can support task-oriented organization of information. In addition, a community semantic web can present semantically related contextual information for each interaction. This information can draw from the history of community interactions and support a better interpretation of the interaction in the context of previous community interactions.

Previous research on coordination in OSS communities [GPS04] has found that developers want improved access to their archives. In interviews with OSS developers from various communities, they found that tools that link related conversational streams are particularly desired, because it would allows conversations to be seen in the context of work artifacts. If we consider each conversation within two related conversational streams to provide better context for the interpretation of the other, then the lack of an activity-centric view is a much more general problem.

Ducheneaut et al. [DW05] reviewed previous research on email and developed a number of recommendations for future directions of email. They suggest that contextual data could be fruitfully integrated into email. Some of the contextual data suggested includes interaction histories of people and results from a search query about a person. In particular, for email in an organization, there is a wealth of data about how work is (or should be) organized, by connecting to organizational charts and documents. This data should be accessible from email, providing a context for interpretation of exchanges. In the situation of OSS communities, we can similarly argue that the wealth of information in previous interactions and in the software code should be accessible from individual interactions to provide a context for interpretation of the discussion.

The Taskmaster [BDH$^+$05, HH97] provides a task-centric view to email and link together information related to a task, such as files and chat logs. Users of the system could explicitly include an artifact in the current task. The system would then present all artifacts semantically related to a task together. In the context of email, it is possible for users to explicitly relate files to messages. This is not applicable in the OSS setting. In particular, Taskmaster does not use archived information to link related files.

Essentially, the problem at hand is a retrieval problem, but we remember far more about documents we use than is evident in retrieval facilities [Lan88]. As discussed by Ducheneaut et al. [DW05], there is reason to believe that what is remembered about messages and interactions includes the meaning of their content, contextual information such as what they look like, what one was doing at the time, associated concurrent events and the time of message receipt or composition in terms of message conversations. However, current retrieval facilities only make use of a fraction of these contextual cues. A community semantic web can support retrieval of artifacts in the community archive through numerous paths, as it can enable a greater number of cues to be encoded and presented. As [Lan88] concludes: information that is logically related to the required memory will not succeed in eliciting recall unless it is also related to the way in which that information was interpreted: we need a richer set of metadata.

OSS community members constantly try to improve their work processes and, although it remains unimplemented, the idea of cross-linking shows up within their discussions too:

<M> for what do you plan?
<H> making things easier for people.
<H> interfaces back into the system from comments and the like. . .
<H> fg. say someone wants to talk about a bug in a forum posting they could do something like: in `<bug number=1>bug #1</bug>` *it says. . .*
<H> and that would automatically turn that into a proper link into the bug tracker.

Identifying Expertise

New members of the community often find it difficult to identify the experts in the community. Previous research [Smi02] has argued for the creation and use of social meta-data for communities. Long-term community members are aware of the experts in a given area by monitoring people's activities in the community over a long period of time. A community semantic web can facilitate this process by aggregating information about people's activities already recorded on the community website. By capturing information about people's activities, the community semantic web can then identify experts on certain topics or areas of code and recommend them to newcomers.

The challenges discussed thus far are primarily true for large and active OSS communities, where there are lots of people and numerous interactions that are difficult to keep track. There are probably far more OSS communities that are small or relatively inactive and do not face such challenges. However, such communities face other challenges, for example that of reaching more people, which is not the focus of this work.

2.2 Semantic Web for Community, Content and Interactions

The Web as a collaboration platform facilitates the communication, interaction and storage of the interactions within the OSS community. It functions essentially as a storage and communication medium. The Semantic Web, by enabling meaningful, processible content, has a very different role to play in OSS communities. Instead of being a passive participant in all these processes, the Semantic Web can be an active entity, a *virtual actor*, working alongside the developers. A virtual actor would interact with other developers, process and recognize various kinds of interactions, suggest actions on the basis of these and remember and bring past interactions to the developers' attention, thus enabling developers to work better together.

2.2.1 Scenario

To illustrate the possibilities offered by the Semantic Web, we present a case study of bug resolution in a given community, OpenACS. We consider bug resolution because it is one of the most complex activities within OSS, involving awareness of the most different kinds of artifacts and people's activities.

In illustrate the role of the Semantic Web as a virtual actor in OSS communities, let us examine a series of interactions associated with a bug in OpenACS, a typical OSS project. The interactions around this bug are fairly typical of bug resolution processes in OSS communities.

OpenACS Bug #1511

Right after a major release of the .LRN/OpenACS software [Ope] in late December 2004, a bug (Bug #1511) was discovered in the File Storage package. This component allowed users to share files with each other. The bug produced a "Content not found" error when users attempted to download files in default folders created by the software. This was a critical bug, since a fundamental functionality of the File Storage functionality was broken.

The bug was first noticed on 21st January 2005 and then again on 12th February 2005. The bug report was opened on 14th February, commented on and a potential fix was posted on the 23rd of February. On 27th February, a duplicate bug was filed by a core developer, Dirk. On 28th February, this bug was recognized as being a duplicate of #1511. Dirk

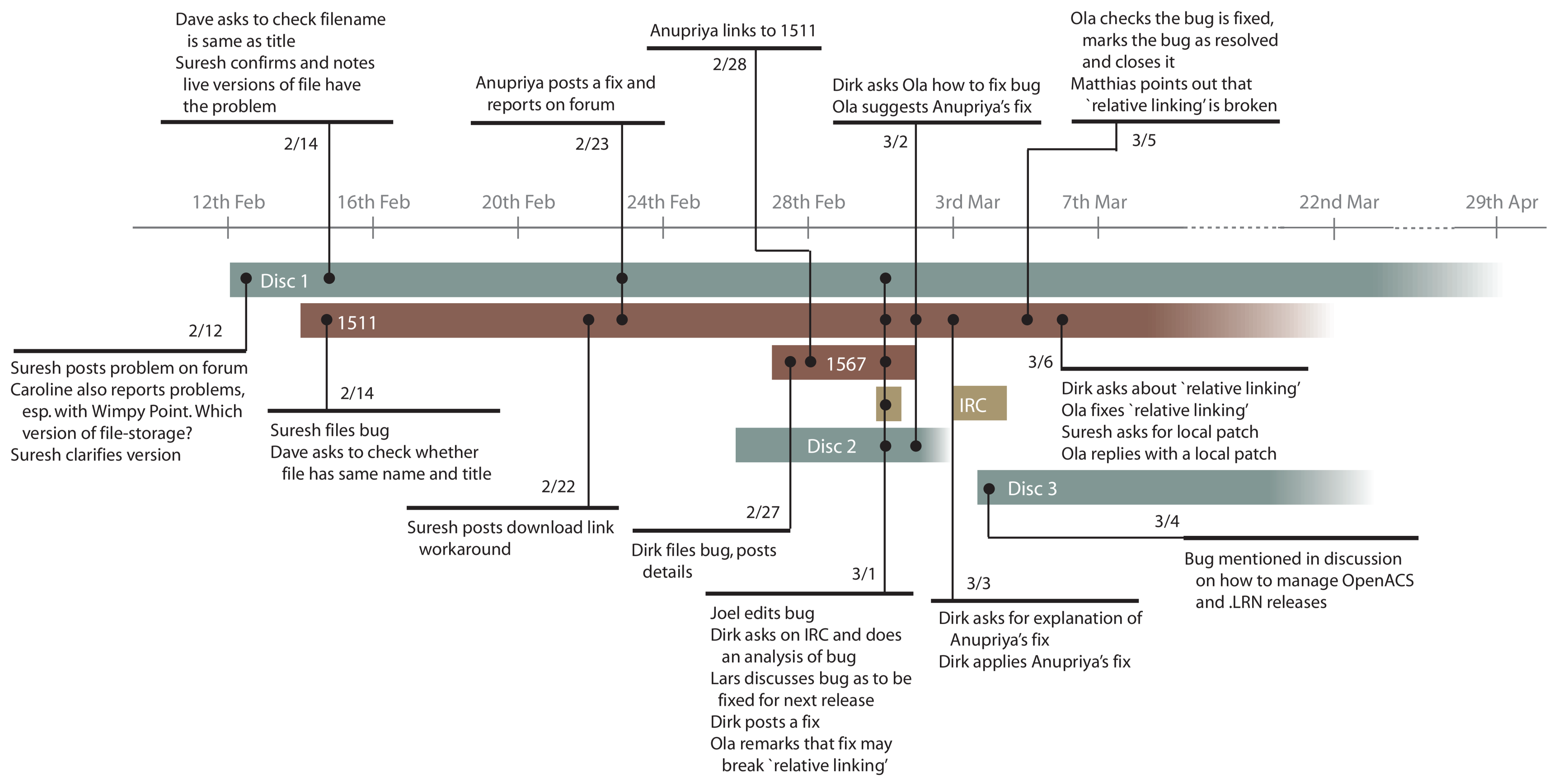

Dave asks to check filename is same as title
Suresh confirms and notes live versions of file have the problem
2/14
Anupriya posts a fix and reports on forum
2/23
Anupriya links to 1511
2/28
Dirk asks Ola how to fix bug
Ola suggests Anupriya's fix
3/2
Ola checks the bug is fixed, marks the bug as resolved and closes it
Matthias points out that `relative linking' is broken
3/5
12th Feb
16th Feb
20th Feb
24th Feb
28th Feb
3rd Mar
7th Mar
22nd Mar
29th Apr
Disc 1
1511
1567
IRC
Disc 2
Disc 3
2/12
Suresh posts problem on forum
Caroline also reports problems, esp. with Wimpy Point. Which version of file-storage?
Suresh clarifies version
2/14
Suresh files bug
Dave asks to check whether file has same name and title
2/22
Suresh posts download link workaround
2/27
Dirk files bug, posts details
3/1
Joel edits bug
Dirk asks on IRC and does an analysis of bug
Lars discusses bug as to be fixed for next release
Dirk posts a fix
Ola remarks that fix may break `relative linking'
3/3
Dirk asks for explanation of Anupriya's fix
Dirk applies Anupriya's fix
3/6
Dirk asks about `relative linking'
Ola fixes `relative linking'
Suresh asks for local patch
Ola replies with a local patch
3/4
Bug mentioned in discussion on how to manage OpenACS and .LRN releases

Figure 2.5: OpenACS Bug #1511

recognized the problem to be critical and immediately began work on it on 1st March. He did not consider the reported fix to be a good solution. He consulted with other developers on the community chat rooms, made a diagnosis and made a fix for the bug, different from the first fix reported. This was reported on a separate discussion, where bugs hindering the latest release of the software were being discussed. His fix was picked up by Ola, a developer who had worked with the File Storage package earlier. On 2nd March, Ola pointed out that Dirks fix might break some required functionality relative linking and suggests using the first fix reported. Dirk fixed the bug as per Olas instructions on 3rd March. Ola checked the fix on 5th March, marks the bug as resolved and closes the bug. The same day, Matthias pointed out that the latest fix breaks relative linking. Dirk asked about what relative linking is on 6th March. At this point, Ola stepped in, fixed the bug and explained his fix. On 9th March, the bug was finally closed as resolved by the submitter. The bug sparked off several discussions about the architecture of the File Storage package, which are still ongoing[3]. The history of this bug is summarized in Figure 2.5.

The bug itself required a relatively simple modification to fix, but it took almost a month-and-a-half to fix. Many of the challenges discussed in Section 2.1.5 showed up here in the lengthy resolution of the bug. To begin with, the information and interactions relating to the bug were located in multiple artifacts with no links. Thus, a duplicate bug was filed. Discussions relating to the bug took place in a discussion forum, which was only linked explicitly later on by one of the developers. As we shall see in the next section, there were several discussions on the same topic as the bug, that were never picked up by the participants of this bug resolution.

The bug was noticed first by someone without expertise in the area of the bug. This led to several iterations of invalid fixes and discussions with an expert, Ola, before the bug was fixed. Bugs that have been fixed by people without expertise in the area of the bug should probably be checked by people with expertise in the relevant area. Not knowing who the experts are unnecessarily prolongs the time until the bug is fixed. Finally, the lack of appropriate documentation also had a role to play in the unfolding of this bug. Dirk did not know what ‘relative linking’ is and there is no available reference documentation to

[3]The latest post to the discussion was on 29th April.

explain what the ‘relative linking’ functionality is and which files it is affected by.

In the next section, we discuss a possible alternate time line of the resolution of the same bug that would be possible if the proposed Semantic Web support were in place. In particular, we shall see how the semantic web can address the challenges we’ve identified.

OpenACS Bug #1511 with a Semantic Web Agent

Given a Semantic Web platform, functioning as a virtual actor, processing the reports and interacting with the developers, the resolution of the bug may have taken quite a different course. It is, of course, difficult to predict how the developers and users might have used the Semantic Web platform, but some possibilities are sketched below.

- As reports of problems or bug symptoms come in, the Semantic Web agent links reports together and to code. In some cases, a person might tell it what the report is about, in others, it might be able to automatically infer from the classification from the text of the report.

  Individual problems can be presented together with other related problem reports. So, problems reported in bug reports and discussion forums do not appear to be isolated, but instead form a large salient group. Furthermore, an actor will not just see related problem reports, but also related interactions in other discussion forums. This will also enable duplicate bugs to be noticed significantly earlier.

  In the case of Bug #1511, the problem reports of Matthew, Suresh, Caroline would appear together with the bug filed by Suresh. When Dirk attempts to file a new bug, the Semantic Web brings these interactions to his attention. Dirk can then look at the forum posts and Suresh’s bug and realize that his bug is a duplicate. Other developers browsing through the discussions or bug reports will notice that there are a number of unresolved discussions associated with either of the posts, pointing to a potential bug that is being overlooked.

- Once the number of related reports has risen above a certain threshold, the Semantic Web agent can alert people who are potential bug-fixers

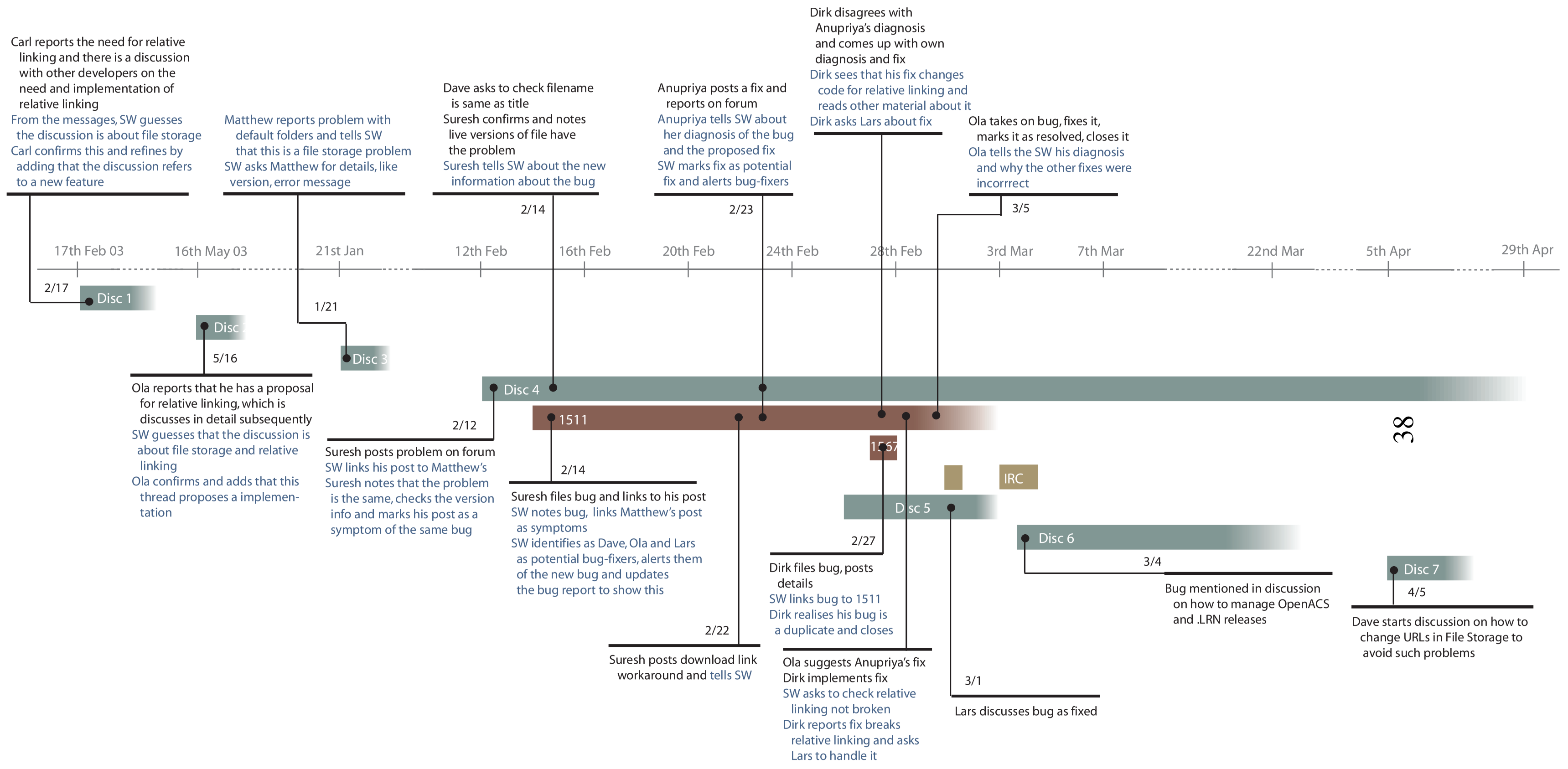

Carl reports the need for relative linking and there is a discussion with other developers on the need and implementation of relative linking
From the messages, SW guesses the discussion is about file storage
Carl confirms this and refines by adding that the discussion refers to a new feature
Matthew reports problem with default folders and tells SW that this is a file storage problem
SW asks Matthew for details, like version, error message
Dave asks to check filename is same as title
Suresh confirms and notes live versions of file have the problem
Suresh tells SW about the new information about the bug
Anupriya posts a fix and reports on forum
Anupriya tells SW about her diagnosis of the bug and the proposed fix
SW marks fix as potential fix and alerts bug-fixers
Dirk disagrees with Anupriya's diagnosis and comes up with own diagnosis and fix
Dirk sees that his fix changes code for relative linking and reads other material about it
Dirk asks Lars about fix
Ola takes on bug, fixes it, marks it as resolved, closes it
Ola tells the SW his diagnosis and why the other fixes were incorrrect
2/14
2/23
3/5
17th Feb 03
16th May 03
21st Jan
12th Feb
16th Feb
20th Feb
24th Feb
28th Feb
3rd Mar
7th Mar
22nd Mar
5th Apr
29th Apr
2/17
Disc 1
Disc
1/21
5/16
Disc 3
Disc 4
1511
1557
IRC
Disc 5
Disc 6
Disc 7
2/12
2/14
2/27
3/4
4/5
2/22
3/1
Ola reports that he has a proposal for relative linking, which is discusses in detail subsequently
SW guesses that the discussion is about file storage and relative linking
Ola confirms and adds that this thread proposes a implemen-tation
Suresh posts problem on forum
SW links his post to Matthew's
Suresh notes that the problem is the same, checks the version info and marks his post as a symptom of the same bug
Suresh files bug and links to his post
SW notes bug, links Matthew's post as symptoms
SW identifies as Dave, Ola and Lars as potential bug-fixers, alerts them of the new bug and updates the bug report to show this
Suresh posts download link workaround and tells SW
Dirk files bug, posts details
SW links bug to 1511
Dirk realises his bug is a duplicate and closes
Ola suggests Anupriya's fix
Dirk implements fix
SW asks to check relative linking not broken
Dirk reports fix breaks relative linking and asks Lars to handle it
Lars discusses bug as fixed
Bug mentioned in discussion on how to manage OpenACS and .LRN releases
Dave starts discussion on how to change URLs in File Storage to avoid such problems

Figure 2.6: OpenACS Bug #1511 with a Semantic Web

The Semantic Web remembers who worked on similar bugs or related code in the past. When someone reports a problem, it can be brought to the attention of the person who is likely to have most knowledge of this code. So, when Suresh filed the bug, the Semantic Web could suggest that Ola might be the person to contact, as Ola worked on the code recently. At the same time, Ola is sent a notification about the bug report filed. When Dirk attempts to file the bug, he too sees that Ola is the person to contact about this bug. Thus, even if he decides to fix it, he can directly ask Ola for advice and help. The responsibility for the bug can then be shared with a domain expert, such as Ola. This also increases the chances of a quicker, more correct diagnosis of the problem.

- The Semantic Web can gather information about the bug and present it to the bug-fixer.

  When either Dirk decides to fix the bug, he can see that the code is associated with several features, including one called 'relative linking'. He knows that if he makes changes there that it might break some of those features. The Semantic Web provides him with history and information about the features and also shows who added them, so Dirk can check with relevant people before making a change.

- When developers report on the resolution of a bug, they also markup and link their comments for the Semantic Web. This way, the bug and other discussions enrich the Semantic Web itself.

There were forum discussions in early 2003 on the implementation of relative linking that were particularly relevant to the bug, but were missed by Dirk, when addressing the problem. Although Ola suspected that the fix might break relative linking, he did not realize that relative linking was indeed broken.

Even searching through the project documentation might not have revealed the discussions. The bug was discussed as part of critical bugs pending resolution before a major release on a discussion forum. Since the bug was a small part of long discussion, formulating the right query to retrieve such information is nontrivial.

The Semantic Web can be thought of as *informating* [Zub88] the OSS community. A technology informates when it not only produces actions, ”but also produces a voice that symbolically renders events, objects and processes so that they become visible, knowable and sharable in a new way. . . . It provides a deeper level of transparency to [underlying productive and administrative] activities that had been either partially or completely opaque.”

2.2.2 Bug Resolution as a Domain for the Semantic Web

In this work, we consider OSS bug resolution as the focal activity to examine the creation of a semantic web. In this section, we explore the characteristics of bug resolution that make it a good candidate activity. Although we have no concrete knowledge of how open source developers approach and fix bugs, there has been much research in software engineering about how software developers, in general, try to resolve bugs [Wel95]. Understanding the software program is crucial for resolving bugs in the program [Wel95, KA86]. When programmers try to understand a program, they essentially try to answer four different kinds of questions [Let86]. Two kinds of questions refer to the semantics of the software that are easily located in the community archive. Namely, ‘What is a given variable or procedure?’ and ‘Why is this functionality implemented in this particular way?’

To answer these questions, programmers often rely on *beacons* [GC91], sets of features that typically indicate the presence of a particular data structure or operation in the source code. Beacons provide the link between the process of verifying hypotheses and the actual source code. So what are beacons? Are they semantic concepts implemented in the source code or structural elements of the code itself? Meaningful identifier names, in particular, procedure and variable names have been shown to aid comprehension and significantly reduce the time required for to understand a program [GC91]. This is because meaningful name are highly indicative of the procedure or variable’s actual function, so programmers can rely on them when trying to understand a program. In addition, procedure names are easily located in the code due to most programming languages’ syntax and formatting conventions [Bro83, Wie86].

A number of cognitive studies of developers trying to understand a program have shown that developers essentially realize plans in their programs [SLPL86, SSO87]. When plans are not implemented in localized regions of code, they hinder the recognition of the plan behind the program [SL86]. Consequently, developers resolving bugs spend up to 60% of their time spent performing simple searches across entire software system [BDS$^+$90]. The need for such searches is the *delocalization* of information in general [Wel97]. More often than not, the information required to understand a section of code is found elsewhere, before or after in the file, in a different file or different directory. As [Wel97] argues, for large software systems, whose source code is spread out over a large number of files in a deep and complex directory structure, even simple searches can become difficult and time-consuming.

To address these issues, Brachman et al. [BDS$^+$90] developed the notion of a Software Information System (SIS). An SIS indexes the software source code and stores relationships that are frequently searched for by software developers during software maintenance. In OSS communities, supporting bug resolution also requires a great understanding of the semantics of the content of developer messages and the context of the project and the actions of developers. Thus, the community semantic web essentially performs the same function as an SIS, but encompasses a greater variety of information, such as bug reports and discussion messages. The community semantic web is therefore much broader in scope.

The envisioned community semantic web (Figure 2.7) thus essentially identifies relevant community resources that could be useful for the resolution of a bug. These resources include artifacts, such as other bug reports, discussion messages, commit log comments and source code files, as well as people who have expertise in the area of the bug.

There are several requirements to realize the community semantic web sketched thus far. To begin with, we need a way to describe the semantics of information on the Web, namely the bug reports, discussion messages, documentation, source code files and commit logs. Given metadata about information in the OSS community artifacts, we then need a way to relate metadata about different artifacts to each other. In other words, the Semantic Web needs to be able to express the relationships between various artifacts through their

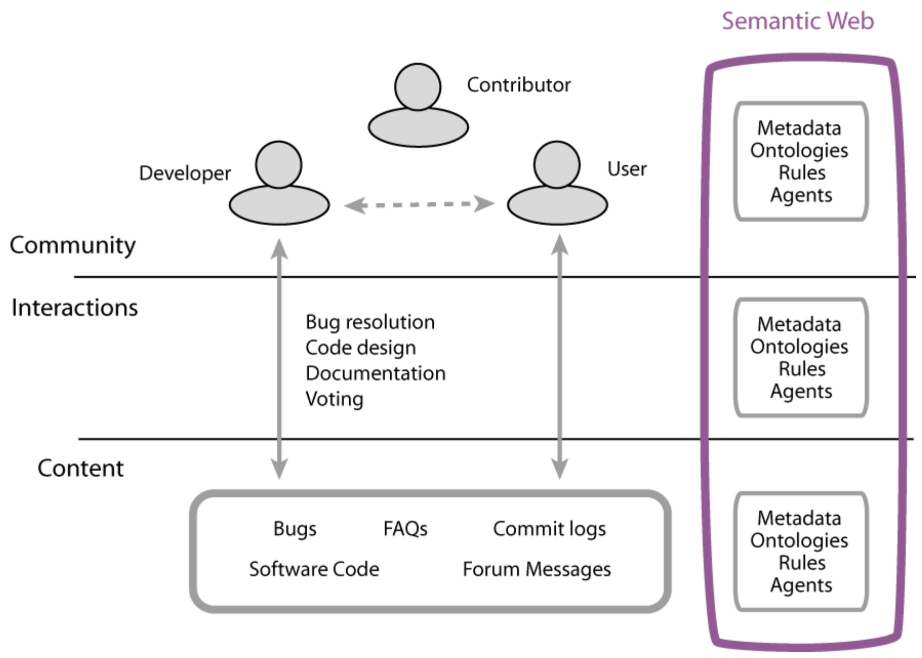


Figure 2.7: The semantic web reflects the three layers of an OSS community

metadata. By expressing metadata and their relationships in a machine-processable manner, we enable the automatic processing, classification and presentation of the artifacts. In addition, in order to interpret the semantics of community interactions, we require natural language processing to process the communication artifacts of a community. Finally, we need information retrieval techniques to retrieve communication artifacts when supporting bug resolution.

2.3 Components of a Semantic Web or What Needs to be Created?

The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation [BLHL01].

The Semantic Web is a major initiative, spearheaded by the W3C Consortium [w3c], to enable Web content "to be shared and reused across application, enterprise and community boundaries" [Sem]. The Web is an excellent medium for the storage and communication of human-comprehensible information. The Semantic Web allows semantic metadata to be attached to Web content, such that the content can be processed and reasoned about by software agents. Currently, the Web is best-suited for human-to-human communication. Supporting human activities on the Web, such as collaborative work, requires that machines also be able to comprehend content on the Web. Instead of training machines to understand natural language, the Semantic Web approach develops languages to express information in a machine-processable form [BL98]. Information on the Web is augmented with documents that explicitly describe relationships between Web resources and contain semantic information intended for automated processing [BLCS99]. While the idea of machine-comprehensible metadata is not new, the Semantic Web is the first large-scale effort to attempt to realize machine-comprehensible metadata for the Web.

A Semantic Web consists of several components: *metadata* to refer to and exchange knowledge about Web resources, *ontologies* that enable the expression of complex relationships between Web resources, *inference engines and rules* that enable new information to be inferred from that already expressed, and *agents*, which harness the above components to perform complex actions for people. In the following, we discuss each of the components of a Semantic Web, the existing standards and what needs to be done for the component in the context of this work.

2.3.1 Metadata

Information about resources on the Web is known as *metadata*, which literally stands for 'data about data'. The metadata for an object or resource can be any descriptive information about the object, including information that is useful to interpret and use the object, such as descriptions of the purpose or utility of the object and representations of the content of the object. For example, object metadata can include the creation date of the object, what the object contains, where (and how) it is stored etc. Metadata associated with files and documents typically comprises the title, the subject, author, publication or modification date, size, format etc.

On the Web, metadata is used to provide information about documents and content items that is typically not displayed on-screen. This information can then be used by software such as search engines, aggregation and presentation applications. For example, search engines on the Web typically weight the keywords in the metadata of a Web page higher than the actual content of Web pages. Metadata can also be used for presentation purposes. A common example is the Cascading Style Sheets (CSS) [CSS] standard, which uses simple metadata for describing the presentation of Web page elements. Metadata is therefore becoming particularly important in XML-based Web applications.

Metadata is typically *structured*, especially in the context of managing electronic data, as on the Web. Structured metadata is typically in the form of labeled fields. For example, metadata about books in a library catalog is usually stored in fields such as 'author', 'title' and 'publisher'. On the Web, structured data is stored in metadata annotation tags. For example, in an HTML document, the document title is marked up as

```
<title>Radha's Homepage</title>
```

where the tags `<title>` and `</title>` are the start and end tags respectively, marking the start and end of the document title.

In the Semantic Web, not all the information with metadata is expressed as text on Web pages. Thus, we require a more generalized way of specifying metadata. Expressing metadata about Web information requires essentially a ability to refer to things on the Web and an ability to make statements about them.

Information on the Semantic Web is referred to through a URI (Uniform Resource Identifier) [URI], which is a Web identifier that can refer to any Web resource. URIs may refer to documents, resources, to people, and indirectly to anything on the Web. A URI is essentially a simple text string, similar to URLs, which specify the location of a Web resource. An example of a URI is:

```
http://infomesh.net/2001/swintro/
```

The second requirement for expressing metadata about Web information is a way to make statements about Web resources. This is the functionality provided by RDF (Resource Description Format) [LS99]. RDF enables people to make statements about Web resources, using a simple language of URI triples. For illustration, consider the RDF/XML [4] fragment below, which states that the URI mentioned in the previous paragraph identifies a Web resource. The Web resource has been created by Sean B. Palmer and is entitled 'The Semantic Web: An Introduction'.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:foaf="http://xmlns.com/0.1/foaf/" >
  <rdf:Description rdf:about="http://infomesh.net/2001/swintro/">
    <dc:creator rdf:parseType="Resource">
      <foaf:name>Sean B. Palmer</foaf:name>
```

[4]RDF is a general-purpose language for representing information in the Web. RDF/XML refers to the RDF information expressed in XML syntax [DB04]. There are several other syntax languages for RDF, such as Turtle [Bec04], a text syntax for RDF, and N-Triples [GDB04].

```
    </dc:creator>
  <dc:title>The Semantic Web: An Introduction</dc:title>
  </rdf:Description>
</rdf:RDF>
```

The lines 1 and 10 define the XML fragment as using RDF tags. Lines 1-3 also declare three namespaces used in the fragment `rdf`, `dc` and `foaf`, which are each declared by the prefix '`xmlns:`'. The namespace `rdf` points to a document located at

`http://www.w3.org/1999/02/22-rdf-syntax-ns#`

This document contains the definition of the RDF tag set. The namespaces `dc` and `foaf` refer to the Dublin Core and FOAF (Friend of a Friend) tagset respectively. The Dublin Core is a metadata standard (set of tags) for describing digital objects (including webpages), usually expressed in XML [Dub04]. For example, the tag `dc:title` and `dc:creator` identifies the title of the Web resource and the person who authored it. Semantic Web tags are usually of the format `[namespace]:[tagname]` and we shall follow the convention throughout this document.

Annotations tags such as these are already common within the Web community; witness the numerous news and weblogging sites offering RSS (RDF Site Summary) [RSS] feeds. RSS is an RDF-based[5] format for syndicating news-like content, such as new items from news sites and posts on personal weblogs. RSS-aware programs, such as news aggregators, check the RSS feed for changes and can react to the changes appropriately, for example by displaying new items or alerting the user to a new post at a particular site. This enables people to monitor a large number of websites with minimal overhead. RSS aggregators are becoming particularly widespread in the weblogging community. Webloggers often keep track of new postings to other people's weblogs. When the posts are made infrequently, though, it is quite tiresome for people to continually check the weblog. With RSS, the news aggregators can poll the weblog and notify when any changes are made.

[5]The RSS specification is not yet a standard. Thus, although one version is based on RDF, there are several other handcrafted languages promoted by certain companies or software [Pil02].

Web entities can also be virtual representations of real entities, such as people. Webloggers[6] are increasingly using the metadata tag set, FOAF (Friend of a Friend) [BM04], which provides a simple set of tags, that enables the description of people, their interests, acquaintances and friends. The following fragment of FOAF[7] illustrates the use of FOAF for specifying people and their acquaintances.

```
<foaf:Person>
  <foaf:name>Leigh Dodds</foaf:name>
  <foaf:knows>
    <foaf:Person>
      <foaf:name>Dan Brickley</foaf:name>
      <rdfs:seeAlso
        rdf:resource="http://rdfweb.org/people/danbri/foaf.rdf"/>
    </foaf:Person>
  </foaf:knows>
</foaf:Person>
```

The above fragment describes a `Person`, whose name is Leigh Dodds and who knows another `Person`, called Dan Brickley. Further information about the `Person` called Dan Brickley can be found at `http://rdfweb.org/people/danbri/foaf.rdf`. A number of tools process and display FOAF data, such as the FOAFnaut [FOAc], which provides a simple social network-like visualisation of FOAF data.

Challenges in Widespread Metadata Use

Although large-scale metadata adoption could usher in a new way of interaction with the Web, there are several challenges that must be overcome before metadata use is widespread. The first issue is that of creating metadata for Web information. There are several tools

[6]A weblog, web log or simply a blog, is a web journal application which contains periodic posts on a common webpage [Web]. A weblogger is an author of a weblog.

[7]excerpted from `http://rdfweb.org/topic/UsingFoafKnows`

that assist user creation of metadata. However, placing the burden of metadata creation on users is clearly an approach that does not scale well for large number of Web resources. In addition, many Web resources are created and changed dynamically. It is therefore natural to require that the metadata for these resources is also created and updated automatically. For some standardized sets of metadata tags, such as RSS and FOAF, there are several tools [FOAa, FOAb, Mov, Blob, Bloa] that facilitate the metadata creation. For example, the automatic creation of an RSS feed for weblog posts is now a standard feature in most weblogging software applications. However, standards with supporting metadata creation tools are few and far between. Most standards, though, lack widely-available tools to help people create metadata.

Recent research has considered the problem of metadata creation. The research can be divided into two areas: one that explores intelligent user support for metadata creation and the other that stresses automatic generation of metadata. Automatic generation of metadata is straightforward if the underlying data to be marked up is highly structured. Such data may stored in a relational database and presented via HTML pages on the Web. In this case, as [HP02] suggest, RDF annotations can be generated dynamically along with the HTML pages. The data model in the database provides an initial tag set for the data. The tag set can later be modified and the RDF generation process appropriately changed. This is essentially the approach followed by the RSS feed generation software.

In some cases, the information to be modeled may be highly structured, but there may be no underlying database or it may not be accessible to the metadata generation process. An alternate approach to automatic annotation [MYR03] thus relies on structural analysis of an HTML page to identify individual data elements of the Web page. Given suitable text in the Web page tags or a tag set hierarchy with a mapping from a set of HTML presentation tags to a label, this method can generate annotations automatically for the most part. Such an approach is particularly well-suited for OSS tool-specific Webpages. There are a small set of bug tracking systems and version control systems popular in the OSS community. Most of these tools have a Web interface that is frequently used by the community. Web pages generated by these tools have low variability in surface structure. They are therefore a prime use case for an automatic annotation approach that uses Web page structure to intelligently generate annotations.

Most of the information on the Web is, however, unstructured. Automatic metadata generation is much more difficult for unstructured information. Research therefore typically focusses on providing semi-automatic annotation. In other words, providing intelligent annotation support to the user. The KIM platform [KPO+03], for example, uses information extraction techniques to extract certain types of information, such as people, places, dates etc. and annotate them automatically. Annotating more general types of information has been explored [BHS02, CDF+00], primarily by using machine learning techniques to learn information extraction for the Web.

In this work, we use a hybrid metadata generation approach. Our approach includes structural analysis of tool-specific HTML pages to extract and markup the structured data contained in the bug tracking system and the CVS repository. Our approach additionally uses information extraction techniques to extract structured domain-specific terms such as software source code elements, from unstructured text. Finally, the approach uses natural language processing to identify potential terms in unstructured natural language text. This combination approach is applied to the historical artifacts of the community (as discussed in Section 2.2). We generate the metadata automatically as far as possible, necessarily trading off some amount of precision to enable automation (see Section 3.3). This is particularly true when extracting potential terms in unstructured natural language text. Automatic generation of metadata, especially induced solely by the data itself is necessarily shallow in its scope and prone to errors. The way these issues play up in the context of this work will be discussed in detail in Section 3.3.3.

A closely related problem to metadata generation is that of metadata use. Unless there are compelling uses of metadata, interest in generating annotations remains low and existing annotations and annotation tools are limited. However, much of the proposed use of metadata and tools to process metadata rely on the existence of lots of annotations, a chicken-and-egg problem. One way out is an approach like Semblog [OTH+04], which provides an integrated end-user oriented environment for gathering, authoring and publishing information and builds on basic metadata tag sets, like RSS and FOAF. Semblog points to the possibilities of the Semantic Web, but focusses primarily on the mechanics of publishing facilitated by the Semantic Web, such as aggregation of content and locating people with similar interests.

In this work, we tie the use of semantic metadata with semi-automatically generated metadata within Dhruv. Changes in the generated metadata change the information interface presented by Dhruv. Thus a basic skeleton of a generate-and-use cycle is already present in Dhruv. The community needs not go through any additional effort to generate or find uses for the metadata. It is only when the ontologies are to be extended or the metadata generation process needs to be modified that the community needs to put in effort. If other OSS communities follow suit, there will already be lots of annotated data that is used within individual communities. Then the potential and need for tools that aggregate data from several communities will arise.

2.3.2 Ontologies and Reasoning

RDF provides a simple datatyping-like model to represent Web content, closely related to the relational database model [BL02]. However, RDF on its own does not make Web information more meaningful to machines, any more than knowing the column names of a relational database might. In order to relate various metadata tags to each other, such as that between a `Book` and an `Author`, we require a mechanism to specify the relationships between metadata tags. This is provided by an *ontology*.

An ontology[8] is controlled vocabulary for the formal description of conceptual objects and relations between them [BJ04]. The objects and their relationships are assumed to exist within some domain of interest. Ontologies can include glossaries, taxonomies and thesauri and complex typing of concepts and relationships, but normally have greater expressivity and stricter rules than these tools. A formal ontology is a controlled vo-

[8]The field of ontologies has its roots in philosophy, where its focus is on the fundamentally different categories of things that can exist, essentially on defining the essence and meaning of being. In computer science, an ontology is the attempt to formulate an exhaustive and rigorous conceptual schema within a given domain, a typically hierarchical data structure containing all the relevant entities and their relationships and rules (theorems, regulations) within that domain. The fundamental role of an ontology is to support knowledge sharing and reuse.

cabulary expressed in an ontology representation language. Ontologies resemble faceted taxonomies but use richer semantic relationships among terms and attributes, as well as strict rules about how to specify terms and relationships. The vocabulary is used to make queries and assertions. Ontological commitments are agreements to use the vocabulary in a consistent way for knowledge sharing. Because ontologies do more than just control a vocabulary, they are thought of as knowledge representation. The oft-quoted definition of an ontology is "the specification of a conceptualization of a knowledge domain" [Gru93].

In the Semantic Web architecture, an ontology gives a shared and precise definition to RDF annotations. Several ontology languages have been developed to define the annotations in semantic markup. The earliest languages were OIL [FvHH$^+$01] and DAML [HvHPS01], followed by DAML+OIL [HM00] and most recently, OWL (Web Ontology Language) [DCvH$^+$02]. OWL has been developed by the W3C Web Ontololgy Working Group and is a W3C Recommendation. OWL builds on the RDF schema and now also uses XML as syntax.

The OWL standard consists of three languages, in ascending expressivity: OWL Lite, OWL DL and OWL Full. OWL Lite and OWL DL are based on a logic framework called description logic. OWL Lite is the least expressive of the three, but this is compensated by the existence of efficient reasoning services for it [HPS03]. OWL DL provides more constructors than OWL Lite and extends the use of some of the constructors in OWL Lite. It is closer to a standard description logic. OWL DL has been carefully designed to keep reasoning decidable, although there are still no known algorithms that can reason over all of OWL DL. OWL Full is the most expressive and most compatible with RDF semantics, but inference in OWL Full is undecidable.

In the following, we focus solely on OWL DL, primarily because it provides the best tradeoff between expressivity and reasoning power. Henceforth, when we refer to OWL, we shall always be referring specifically to OWL DL, unless indicated otherwise. OWL Lite was too restrictive for our use, mainly because it does not allow building classes through intersection, union and complement. To illustrate the use of OWL, consider the following OWL fragment [SWM04]. It defines two atomic classes in a vocabulary, `Country` and `Person`. Furthermore, it defines `Student` as being a subclass of

Person.

```
<owl:Class rdf:ID="Country"/>
<owl:Class rdf:ID="Person"/>

<owl:Class rdf:ID="Student">
  <rdfs:subClassOf rdf:resource="#Person" />
</owl:Class>
```

Having defined these classes, the following fragment describes an individual USA as being a member of Country.

```
<Country rdf:ID="USA" />
```

In addition to classes and subclasses, OWL can be used to describe relations between classes (e.g. disjointness), cardinality, equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes [MvH04]. These are discussed in greater detail in the next section.

Formal definition of OWL Ontologies

OWL has a formal basis in description logic[BCM+03]. It is difficult to determine the requirements for a web ontology language given the paucity of existing applications to learn from, however, description logic is thought to have the right balance of expressivity and efficiency for the Web [BHS05]. Description logics are a family of knowledge representation formalisms that represent the knowledge of an application domain through a knowledge base comprising a terminology and a world description. OWL DL is based on the description logic $\mathcal{SHOIN}(\mathbf{D})$ [HPS03]. Formally, a description logic knowledge base (KB) comprises two components, the *TBox* and the *ABox*. The TBox specifies the *terminology* or vocabulary of an application domain and the ABox contains *assertions* about individuals in terms of the terminology. An OWL ontology can contain both TBox axioms and ABox assertions.

The terminology defines relevant concepts of the domain (such as the classes Country and Person above). In the following, we use the terms 'concept' and 'class' interchangeably. More formally, the TBox consists of concepts, which denote a set of individuals, and roles, which denote a binary relationship between individuals [BCM+03]. Concepts and roles can be atomic or complex. Atomic concepts and roles are drawn from two disjoint alphabets of symbols. Complex descriptions of concepts can be built by using concept constructors such as conjunction, disjunction and negation over atomic and complex concepts. For example, the term $C \sqcap D$ is the intersection or conjunction of concepts C and D and denotes the set of all individuals that belong to both C and D. All concepts expressions have a set-theoretic interpretation. The term $C \sqcap D$ is therefore equivalent to the first-order logic statement $C(x) \wedge D(x)$, where the variable x ranges over all the individuals in the interpretation and $C(x)$ is true for all individuals that belong to the concept C. Such descriptions can be assigned names in the TBox, amounting to a concept definition. For instance, a Parent may be defined as the union of the concepts Mother and Father [BCM+03]:

$$\texttt{Parent} \equiv \texttt{Mother} \sqcup \texttt{Father}$$

In OWL DL, a concept can also be defined by enumerating its instances; this is known as an *enumerated class*.

Similarly, new roles can be defined using the inverse role constructor. For example, $\texttt{Male} \equiv \texttt{Female}^{-}$. One of the more unusual features of description logics are *value restrictions*, which describe relationships between concepts. For example, a value restriction of the form $\forall R.C$ states that all individuals that are in relationship R with a given concept are instances of class C. The individuals in relationship R with a given concept are also known as *role fillers*. In other words, the value restriction $\forall R.C$ states that all fillers of role R must be members of concept C. The existential restriction $\exists R.C$ is analogously defined. Together, these role restrictions allow one to define a Parent as someone who has a child (hasChild), who is a Person and all who children are members of Person:

$$\texttt{Parent} \equiv \exists\ \texttt{hasChild.Person} \sqcap \forall\ \texttt{hasChild.Person}$$

Another kind of role restrictions are *number restrictions*, which restrict the cardinality of role fillers. To illustrate, the definition

$$\texttt{MotherWithManyChildren} \equiv \texttt{Mother} \sqcap \geqslant 3\texttt{hasChild}$$

describes a `MotherWithManyChildren` as being a `Mother` with at least 3 children.

The world description or ABox uses concepts from the terminology to assert concept and role properties of individuals occurring in the domain. The former are typically called *membership assertions* or *concept assertions*. For example,

$$(\texttt{Female} \sqcap \texttt{Person})(\texttt{Jane})$$

describes an individual `Jane` as being a female person. Similarly, *role assertions* such as

$$\texttt{hasChild}\,(\texttt{Jane}, \texttt{Julie})$$

state that the individual `Jane` has a child, the individual `Julie`.

The OWL DL description logic has a set-theoretic semantics, such that concepts are interpreted as a set of individuals and roles as sets of pairs of individuals. Atomic concepts are therefore interpreted subsets of the interpretation domain and complex concepts are interpreted as the set of all individuals satisfying the concept description. The interpretation domain may be infinite, which distinguishes OWL DL from database modeling languages, which always assume a finite domain, namely the objects in the database. Another distinguishing feature of description logics is the *open-world assumption*, as explained in [BCM$^+$03]:

> While a database instance represents exactly one interpretation, namely the one where classes and relations in the schema are interpreted by objects and tuples in the instance, an ABox represents many different interpretations, namely all its models. As a consequence, absence of information in a database instance is interpreted as negative information, while absence of information in an ABox only indicates lack of knowledge.

Thus, the *open-world* semantics of OWL is quite different from the *closed-world* semantics of databases and affects the way queries is answered. In addition, OWL deviates from many description logic semantics by not making the *unique names assumption*. Thus, distinct individual names may refer to the same individual. Objects may explicitly be equated using the construct `sameAs`. Objects can also be explicitly distinguished using the `differentFrom` and `AllDifferent` constructs. The rationale behind this assumption is there are several ways to refer to the same object on the Web.

Reasoning and Inference

Reasoning is a mechanism to infer implicitly represented knowledge from the knowledge explicitly contained in a knowledge base [BCM$^+$03]. As a description logic, OWL has been designed to support classification of concepts and individuals, a common inference in many applications of intelligent information processing systems.

The classification of concepts determines subclass-superclass relations between concepts, also known as *subsumption* and expressed as $C \sqsubseteq D$. Checking subsumption is the problem of checking whether C always denotes a subset of the set denoted by D. Another basic reasoning task on concept expressions is *concept satisfiability*, whether a concept C always denotes the empty set $\emptyset$. This problem can be reduced to checking whether C is subsumed by the empty concept. Other TBox reasoning tasks are *equivalence* and *disjointness* of concepts.

The basic inference task for an ABox is *instance checking*, whether a given individual is an instance of a specified concept [BCM$^+$03]. Other ABox reasoning tasks build upon instance checking. These are *consistency*, whether each concept in the knowledge base has at least one instance; *realization*, which finds the most specific concept an individual object is an instance of; and *retrieval*, which finds all instances of a given concept in the knowledge base.

The primary function of the reasoner is to return answers to queries. Reasoners typically use the structure of the class definitions (structural subsumption) [BPS94] or tableau-based methods [SSS91] for inference [BCM$^+$03]. Tableau reasoners are the currently among the most efficient description logic reasoners. In particular, Racer [HM01] and FaCT [Hor98] are two of the most well-recognized description logic reasoners for OWL.

In this work, we primarily used Racer because of its efficiency.

A number of other tools are available for OWL. There are editors like Protege [NSC$^+$01] and SWOOP [KPH05] to create ontologies in OWL, ontology validators such as OWL Validator [Val] that ensure that an ontology is a valid OWL ontology and APIs such as the OWL API [OWL] and Jena [Jen] to provide programming access to OWL ontologies.

Rules

Description logics allow knowledge to be expressed in terms of concepts and world descriptions. However, there are more general kinds of knowledge that cannot be expressed within description logics. A simple example is "if an individual is known to be an instance of C, then it is also known to be an instance of D" [BCM$^+$03]. This is an example of a *rule*. Rules can trigger changes in the knowledge base. For example, given a rule "All students eat only junk food", if the knowledge base knows that Peter is a student, then the processing of the above rule asserts in the knowledge base that Peter eats only junk food.

There have been a number of proposals to augment OWL knowledge bases with rules, such as the Semantic Web Rule Language (SWRL) [HPSB$^+$03] and OWL Rules [HPS04], SWRL being the more established of the two. SWRL enables an OWL knowledge base to be combined with Horn logic rules. A SWRL rule is in the form of an implication form between an antecedent and a consequent, such that whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold. There are a number of rule engines that have been developed or adapted for the Semantic Web, such as Jena [Jen] and Jess [KR03].

Rules for the Semantic Web is however an area that is still in flux and is far from mature. Due to this, we did not use Semantic Web rules directly, but hard-coded rule-like reasoning within Dhruv. Once Semantic Web rules are established, it would be simple to switch over to using them.

Challenges to Semantic Web Ontologies

> ”The real power of the Semantic Web will be realized when people create many programs that collect Web content from diverse sources, process the information and exchange the results with other programs. The effectiveness of such software agents will increase exponentially as more machine-readable Web content and automated services (including other agents) become available.” [BLHL01]

Despite the comprehensive ground work, there remain several challenges and unresolved issues in using ontologies and reasoning for the Semantic Web.

There is lingering discussion [dBLPF05] about the right kind of ontology language for the Semantic Web and whether there should even just be a single one. Proponents of an alternate ontology language tend to question the appropriateness of the open-world assumption for Semantic Web and prefer DB-like answers. In particular, they favor logic programming based languages, such as F-Logic [KLW95] and Flora [Flo]. Logic programming languages are typically less expressive than description logic based languages, but can easily handle large numbers of instances. [GHVD03]. In this work, we opted for the most mature contender for ontology languages, OWL. One of the latent questions we seek to explore in this work is the expressivity of ontologies required for a community semantic web.

The capabilities, syntax and semantics of Web ontologies have been researched for a while and there are reasonably robust solutions in existence. However, there has been much less focus on actual domain-specific ontologies, which hinder the development of a full Semantic Web. There are a few large ontologies that have been expressed in OWL, particularly in the biology domain like the NCI Thesaurus [NCI] and the Gene Ontology [GO]. In general, though, the lack of sufficient domain-specific ontologies is a major hurdle besetting the greater acceptance of the Semantic Web.

In particular, there are no existing ontologies for online communities, besides the FOAF ontology for describing social networks. Therefore, as part of this work we need to create ontologies for the OSS community. The ontologies will form one of the contributions of this work.

2.3.3 Related Work

The individual components of a Semantic Web, such as metadata, ontologies, rules, reasoning engines and agents, have been developed and refined over the past few years and are fairly well-understood at this stage. However, the Semantic Web is still at an early stage. It is not here yet. As in the case of the Web, its future depends largely on the kinds of compelling applications that exist.

There are a few applications of the Semantic Web variously to email and personal information [MEG$^+$03], personal information management [QHK03] and for browsing through marked-up Web content [DDM03] and e-learning. Haystack [QHK03]is an end-to-end Semantic Web application for personal information management. The concept is similar to that of the community semantic web, but it focusses on managing the personal information of a single user. Magpie [DDM03] is a browser that exposes the semantic metadata of a Webpage to the end-user through right-clicks. Mangrove [MEG$^+$03] is a 'semantic email system'. By augmenting email with simple metadata (e.g. yes/no for replies), an agent can process multiple emails and summarize the replies. Mangrove is a relatively lightweight extension of an existing communication mechanism that is commonly associated with the Web. It is possible that the data collated by the Mangrove semantic email agent be presented as semantic information on the Web. However, to our knowledge, that has not been explored.

Most of the above related work (save Haystack) assumes the existence of domain ontologies and present tools to process information on their basis. However, there is a research gap in the application of Semantic Web to a specific domain, especially a dynamic domain constantly changing due to interactions of the Web community. Although individual components of the Semantic Web are well-developed, there is a need for more application knowledge of the Semantic Web. How can the Semantic Web be used by a Web community? What does it take to create a Semantic Web for an existing community?

How does the introduction of the Semantic Web change the interactions of the community? How do community interactions enrich the Semantic Web itself? The proposed work attempts to address this research gap.

Almost two decades ago, Otis Elevators developed Otisline [MS86], a computer application developed to improve Otis Elevators' responsiveness to its service customers. Otisline maintained a database of service requests, which was updated to maintain data on actions necessary to repair elevators. This database was used to reduce callback time to a third of the industry average and additionally, allowed the management to allocate resources to locations with recurring problems and by engineering to spot trends that indicated elevator design problems. When Otisline received a service request, it would log all the important information and dispatch a service mechanic to make the call. Before taking the call, the mechanic could quickly view the problem and all the associated information in the database. After resolving the problem, the mechanic would report to Otisline about the steps taken to repair the elevator. Similarly, the Semantic Web can match bugs to people and assist during the bug resolution phase by bringing relevant experiences to the fore.

An example closer to the open source software development context is Hipikat [CM03]. Hipikat is an Eclipse plug-in [Ecl], which builds a group memory out of all the artifacts in an open source project. Then, someone viewing the code or bugs is presented with information that is considered similar using information retrieval techniques. Hipikat differs from the approach presented here in that it focusses on the single-developer process of solving a bug. My focus is on the multi-developer bug resolution processes that take place beyond the actual fixing of the bug.

Another research similar in spirit is the Living Memory [Cas00] [Cas] project, which explored the role of memory in physical communities, who live and work in a particular neighborhood. The project aimed to provide community members with means to capture, share and explore their collective memory as well as develop mechanisms for people to update their collective memory. The research took a vastly different approach by building physical devices that enabled people to capture and share memories associated with a physical location. In this work, the emphasis is on constructing a logical interface for people to access and interact with their shared memory.

Claimaker [LUM$^+$02] explicitly models the rhetorical relations between claims in related papers, thus modeling readers' interpretations of the core content of papers. This allows for inter-document queries, but requires a lot of work from people to explicitly markup the claims of papers and whether it supports of challenges. Research shows that people are typically loath to do this. In this work, we take a mostly-automatic approach where system tries to learn as much as it can in the background from people's actions and statements.

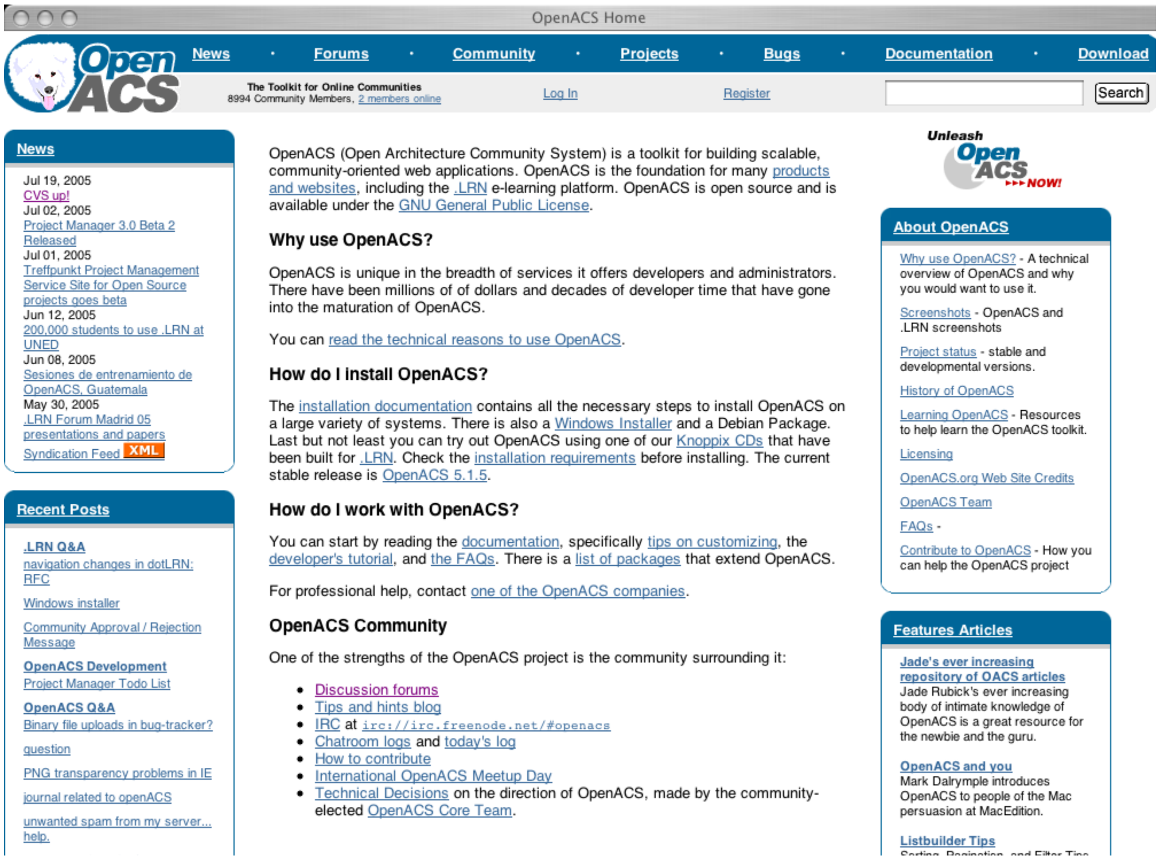
McBride argues in his paper [McB02] that successful Semantic Web adoption requires that practical applications, simple and error-tolerant software, open source infrastructure and applications that can be developed now. Most of these requirements are covered by the proposed work.

Ultimately, something like [LTT$^+$03] is required to provide a collaborative, community-oriented ontology server in an open environment. The larger OSS community itself needs to have control of its ontologies and manage them on its own. This does require some expertise on the part of the community. It may be that there will be special roles for knowledge modeling, just as many OSS projects now have some people that perform the graphic design work for the community.

2.4 Exploration Context: The OpenACS/dotLRN Community

To guide the exploration and creation of a Semantic Web for OSS communities, we focus on supporting a single OSS community, the OpenACS/dotLRN open source software community [Ope]. The OpenACS/dotLRN community has formed around the open source OpenACS (Open Architecture Community System) toolkit, for building scalable, community-oriented web applications. OpenACS provides the foundation for many web applications, including the open source dotLRN e-learning platform [Dot], and many websites, including Greenpeace [gre]. The dotLRN platform has been adopted by several major school and universities, including the MIT Sloan School of Management and the

University of Heidelberg. The dotLRN community is fairly large and contributes a significant amount of code back into the OpenACS community. Therefore, we refer to both communities together as the OpenACS/dotLRN community.


OpenACS Home
Open ACS
News · Forums · Community · Projects · Bugs · Documentation · Download
Log In
Register
Search
Unleash Open ACS NOW!

The Toolkit for Online Communities
8994 Community Members, 2 members online

News

Jul 19, 2005
CVS up!
Jul 02, 2005
Project Manager 3.0 Beta 2 Released
Jul 01, 2005
Treffpunkt Project Management Service Site for Open Source projects goes beta
Jun 12, 2005
200,000 students to use .LRN at UNED
Jun 08, 2005
Sesiones de entrenamiento de OpenACS, Guatemala
May 30, 2005
.LRN Forum Madrid 05 presentations and papers
Syndication Feed XML

Recent Posts

.LRN Q&A
navigation changes in dotLRN: RFC

Windows installer

Community Approval / Rejection Message

OpenACS Development
Project Manager Todo List

OpenACS Q&A
Binary file uploads in bug-tracker?

question

PNG transparency problems in IE

journal related to openACS

unwanted spam from my server... help.

OpenACS (Open Architecture Community System) is a toolkit for building scalable, community-oriented web applications. OpenACS is the foundation for many products and websites, including the .LRN e-learning platform. OpenACS is open source and is available under the GNU General Public License.

Why use OpenACS?

OpenACS is unique in the breadth of services it offers developers and administrators. There have been millions of of dollars and decades of developer time that have gone into the maturation of OpenACS.

You can read the technical reasons to use OpenACS.

How do I install OpenACS?

The installation documentation contains all the necessary steps to install OpenACS on a large variety of systems. There is also a Windows Installer and a Debian Package. Last but not least you can try out OpenACS using one of our Knoppix CDs that have been built for .LRN. Check the installation requirements before installing. The current stable release is OpenACS 5.1.5.

How do I work with OpenACS?

You can start by reading the documentation, specifically tips on customizing, the developer's tutorial, and the FAQs. There is a list of packages that extend OpenACS.

For professional help, contact one of the OpenACS companies.

OpenACS Community

One of the strengths of the OpenACS project is the community surrounding it:

- Discussion forums
- Tips and hints blog
- IRC at irc://irc.freenode.net/#openacs
- Chatroom logs and today's log
- How to contribute
- International OpenACS Meetup Day
- Technical Decisions on the direction of OpenACS, made by the community-elected OpenACS Core Team.

About OpenACS

Why use OpenACS? - A technical overview of OpenACS and why you would want to use it.

Screenshots - OpenACS and .LRN screenshots

Project status - stable and developmental versions.

History of OpenACS

Learning OpenACS - Resources to help learn the OpenACS toolkit.

Licensing

OpenACS.org Web Site Credits

OpenACS Team

FAQs -

Contribute to OpenACS - How you can help the OpenACS project

Features Articles

Jade's ever increasing repository of OACS articles
Jade Rubick's ever increasing body of intimate knowledge of OpenACS is a great resource for the newbie and the guru.

OpenACS and you
Mark Dalrymple introduces OpenACS to people of the Mac persuasion at MacEdition.

Listbuilder Tips

Figure 2.8: The OpenACS Homepage

2.4.1 Community

The OpenACS/dotLRN community is particularly suitable as a context for this exploratory research for several reasons, social and technical. First of all, it is a large, active and diverse community, in comparison to other open source software communities, with around

1700 members, although there are less than 100 active members at any time. The awareness burden is therefore particularly acute here for community members as they must keep abreast of a great variety of artifacts and the activities of numerous community members. The community includes consultants, web developers and administrators, students and through dotLRN, several university staff too. Some members of the community, especially the university staff, are paid to work on OpenACS/dotLRN, whereas others, especially consultants, work on OpenACS/dotLRN to improve their own Web products. Due to the community's close link with research and academia, it is very receptive to research conducted on their platform. The OpenACS platform itself originates from the doctoral dissertation research of one of its founding members.

The OpenACS[9] platform is technically suitable for this research, because it incorporates software development tools, such as the OpenACS Bugtracker, which integrates different aspects of their development environment. The homegrown BugTracker, for example, is an issue tracking software, but unlike many other open source bug tracking software, also links bugs with the patches that fixed them. In addition, the OpenACS community requires people to register at the community website, before participating in community activities. Thus, community members have mostly unique identities on the OpenACS website, making it easy to track the activities of individual community members. In other open source software communities without such a registration requirement, determining and collating people's multiple identities is itself an arduous first step of conducting research.

2.4.2 Activity

In addition to choosing an OSS community for our exploration, we need to also focus on a specific activity in OSS communities, in this case, *bug resolution*. As discussed in the previous chapter, bug resolution is one of the most complex activities in an OSS community and indeed in software development itself. It requires community members to

[9]Henceforth, OpenACS/dotLRN will be abbreviated as OpenACS. Thus, by mentioning the OpenACS community, we are actually referring to the OpenACS/dotLRN community.

possess a solid understanding of the software source code, to be aware of the activities of other community members, as well as maintain awareness of ongoing and past community discussions and bug reports. All of these could potentially inform and influence the resolution of a bug. Thus, bug resolution could benefit from the Semantic Web's ability to bring semantically related artifacts together.

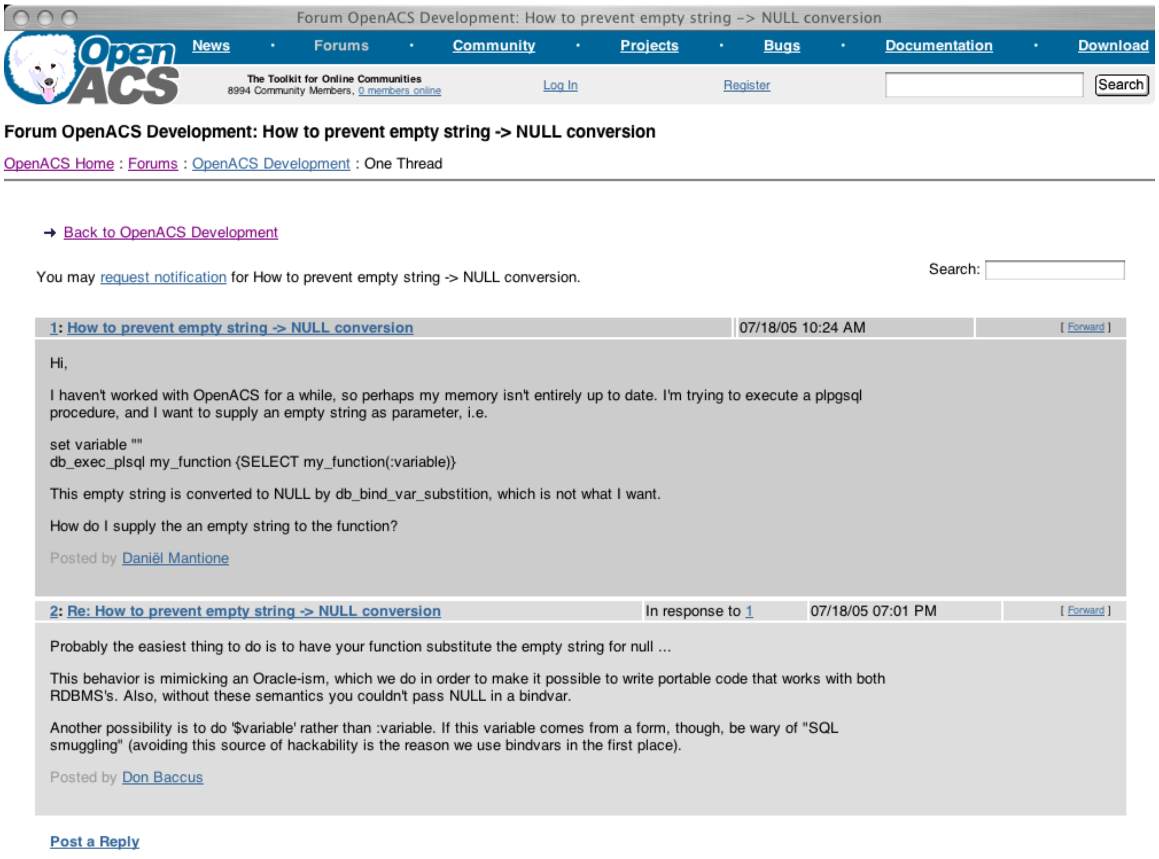
Supporting bug resolution is also likely to be beneficial to the OSS community for several reasons. To begin with, timely and correct bug resolution is crucial to maintaining the quality of the software being produced. Community members take great pains to report bugs, diagnose and fix them as soon as they can. This keeps the software usable and reflects well on the community. However, bug resolution is also one of the least exciting activities. Community members are primarily volunteers, who work on OpenACS in their spare time. Their primary interest is in adding new functionality, so bug resolution is often drudgery to them. By supporting the bug resolution process and helping community members resolve bugs, Dhruv can be particularly useful to open source software communities. It provides a good entry point for Semantic Web systems into online Web community work.

2.4.3 Community Artifacts

The OpenACS community makes use of various kinds of artifacts. The community conducts its work via Web-based forums (Forums), chat rooms, a bug, issues and patches tracker (BugTracker), a source code repository and change tracker (CVS) and source code documentation. The OpenACS website links to all of these. In order to scope of data sources to be examined, we restrict ourselves to information contained in the Forums, Bugtracker and CVS. We next discuss each of these briefly in the context of OpenACS.

OpenACS Forums

The OpenACS Forums provide a means for threaded discussions in the community. The OpenACS Forums contains several topic-based forums, such as `OpenACS Development`, for developers working on the OpenACS code base, and `OpenACS Q&A`, an OpenACS


Forum OpenACS Development: How to prevent empty string -> NULL conversion
News · Forums · Community · Projects · Bugs · Documentation · Download
The Toolkit for Online Communities
8994 Community Members, 0 members online
Log In
Register
Search

Forum OpenACS Development: How to prevent empty string -> NULL conversion

OpenACS Home : Forums : OpenACS Development : One Thread

→ Back to OpenACS Development

You may request notification for How to prevent empty string -> NULL conversion.

Search:

1: How to prevent empty string -> NULL conversion 07/18/05 10:24 AM [Forward]

Hi,

I haven't worked with OpenACS for a while, so perhaps my memory isn't entirely up to date. I'm trying to execute a plpgsql procedure, and I want to supply an empty string as parameter, i.e.

```
set variable ""
db_exec_plsql my_function {SELECT my_function(:variable)}
```

This empty string is converted to NULL by db_bind_var_substition, which is not what I want.

How do I supply the an empty string to the function?

Posted by Daniël Mantione

2: Re: How to prevent empty string -> NULL conversion In response to 1 07/18/05 07:01 PM [Forward]

Probably the easiest thing to do is to have your function substitute the empty string for null ...

This behavior is mimicking an Oracle-ism, which we do in order to make it possible to write portable code that works with both RDBMS's. Also, without these semantics you couldn't pass NULL in a bindvar.

Another possibility is to do '$variable' rather than :variable. If this variable comes from a form, though, be wary of "SQL smuggling" (avoiding this source of hackability is the reason we use bindvars in the first place).

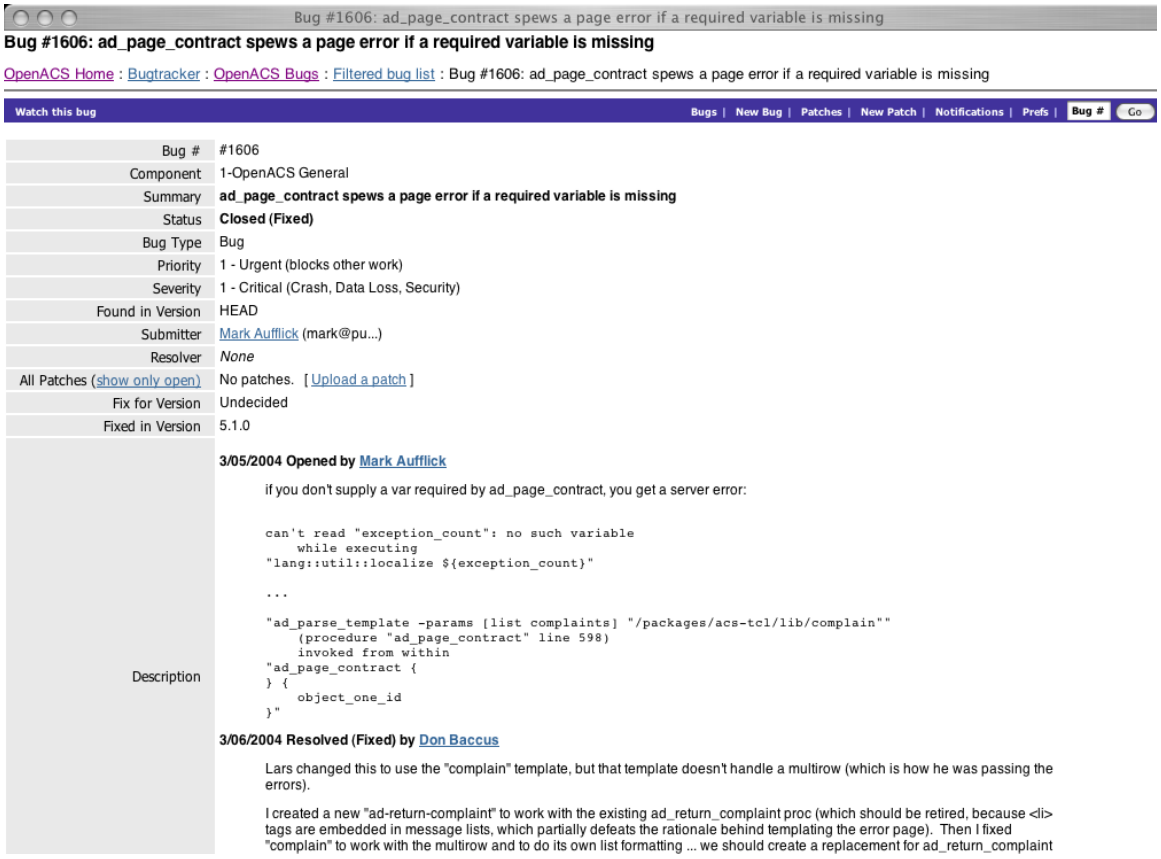
Posted by Don Baccus

Post a Reply

Figure 2.9: Screenshot of an OpenACS Forums discussion

user support forum. Each forum has a number of threads and each thread has multiple messages. Figure 2.9 shows the screenshot of an OpenACS forum thread with two messages in the `OpenACS Development` forum. The subject of the thread is “How to prevent empty string -> NULL conversion”. In the following, we will often refer to OpenACS Forums discussion threads simply as ‘forum discussions’.

OpenACS Bugtracker


Bug #1606: ad_page_contract spews a page error if a required variable is missing
Watch this bug
Bugs | New Bug | Patches | New Patch | Notifications | Prefs | Bug # Go

Bug #1606: ad_page_contract spews a page error if a required variable is missing

OpenACS Home : Bugtracker : OpenACS Bugs : Filtered bug list : Bug #1606: ad_page_contract spews a page error if a required variable is missing

Bug #	#1606
Component	1-OpenACS General
Summary	**ad_page_contract spews a page error if a required variable is missing**
Status	**Closed (Fixed)**
Bug Type	Bug
Priority	1 - Urgent (blocks other work)
Severity	1 - Critical (Crash, Data Loss, Security)
Found in Version	HEAD
Submitter	Mark Aufflick (mark@pu...)
Resolver	*None*
All Patches (show only open)	No patches. [Upload a patch]
Fix for Version	Undecided
Fixed in Version	5.1.0
Description	

3/05/2004 Opened by Mark Aufflick

if you don't supply a var required by ad_page_contract, you get a server error:

```
can't read "exception_count": no such variable
    while executing
"lang::util::localize ${exception_count}"

...

"ad_parse_template -params [list complaints] "/packages/acs-tcl/lib/complain""
    (procedure "ad_page_contract" line 598)
    invoked from within
"ad_page_contract {
} {
    object_one_id
}"
```

3/06/2004 Resolved (Fixed) by Don Baccus

Lars changed this to use the "complain" template, but that template doesn't handle a multirow (which is how he was passing the errors).

I created a new "ad-return-complaint" to work with the existing ad_return_complaint proc (which should be retired, because <li> tags are embedded in message lists, which partially defeats the rationale behind templating the error page). Then I fixed "complain" to work with the multirow and to do its own list formatting ... we should create a replacement for ad_return_complaint

Figure 2.10: Screenshot of an OpenACS Bug Report

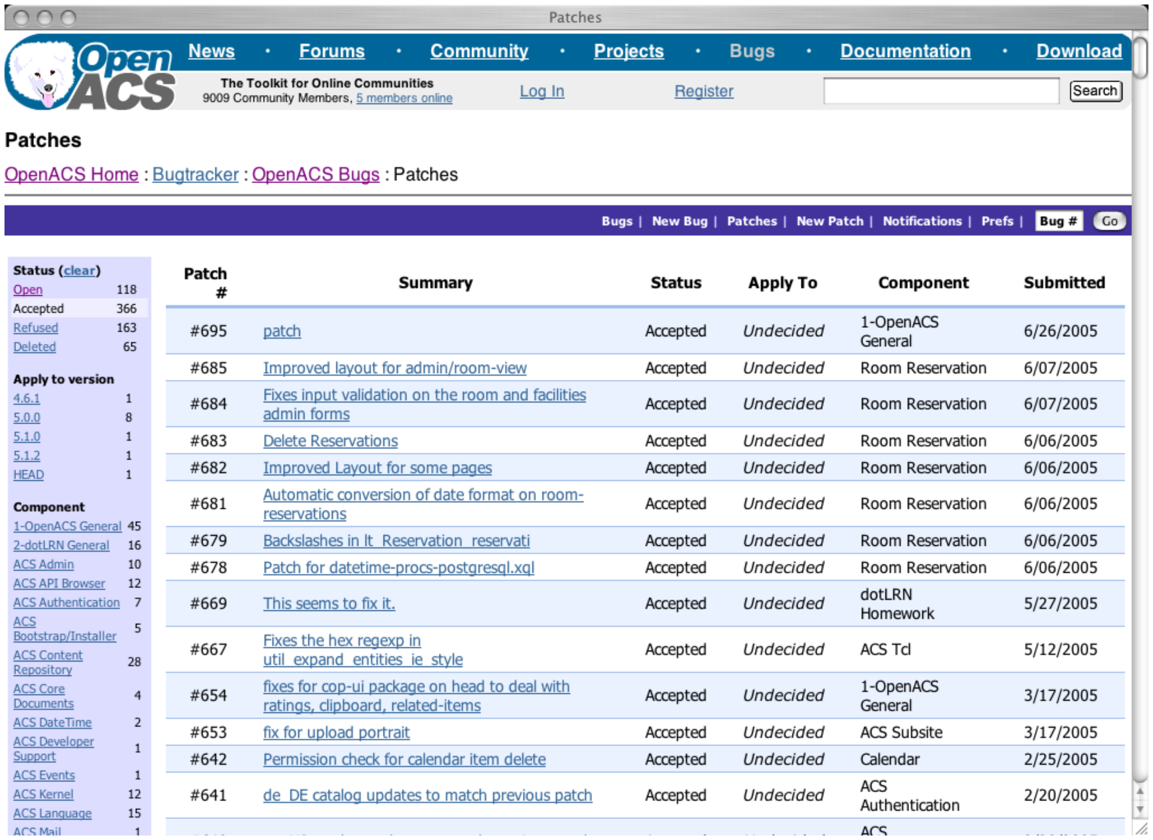
The OpenACS Bugtracker allows all attributes of a bug report and the discussion around a bug to be gathered in a single bug report. An example of a bug report in the

OpenACS Bugtracker can be in Fig. 2.10. The upper section of the bug report lists the attributes of the bug. These attributes are discussed in detail in Section 3.2.2, when the bugs ontology is presented. The lower section of the bug report, titled‘Description’ is essentially a unthreaded discussion specifically around the bug report. In the example shown in the figure, the discussion has two bug report messages. Each bug report message is associated with a date on which the message was posted, and an action, such as ”Opened” or ”Resolved (Fixed)”. This allows community members to perform an action on the bug report and leave a message explaining the action.

An OpenACS patch report looks similar to an OpenACS bug report. Fig. 2.11 shows a screenshot of the overview page of the OpenACS Bugtracker patch reports. OpenACS Bugtracker bug reports have a similar page solely for bug reports.

OpenACS CVS repository

The OpenACS CVS repository has a Web interface that allows people to browse the CVS repository through a Web browser. A screenshot of the OpenACS CVS Web interface is shown in Figure 2.12. The figure shows the commit logs for the file `file.adp` in the package `file-storage`. The commits to the file are sorted, with the most recent commit on top. Each commit includes a brief commit log message which explains the changes made by the commit to the OpenACS code base. Each commit log also has links that allow people to view the version of the file after the commit has been made. This allows community members to view the modifications committed directly from the Web browser without first downloading a local copy of the file. The Web interface to the CVS repository is an important tool for any community member who needs to examine the source code. It is therefore often referred to in the bug reports and forum discussions.


Patches
News · Forums · Community · Projects · Bugs · Documentation · Download
The Toolkit for Online Communities
9009 Community Members, 5 members online
Log In
Register
Search
Bugs | New Bug | Patches | New Patch | Notifications | Prefs | Bug # Go

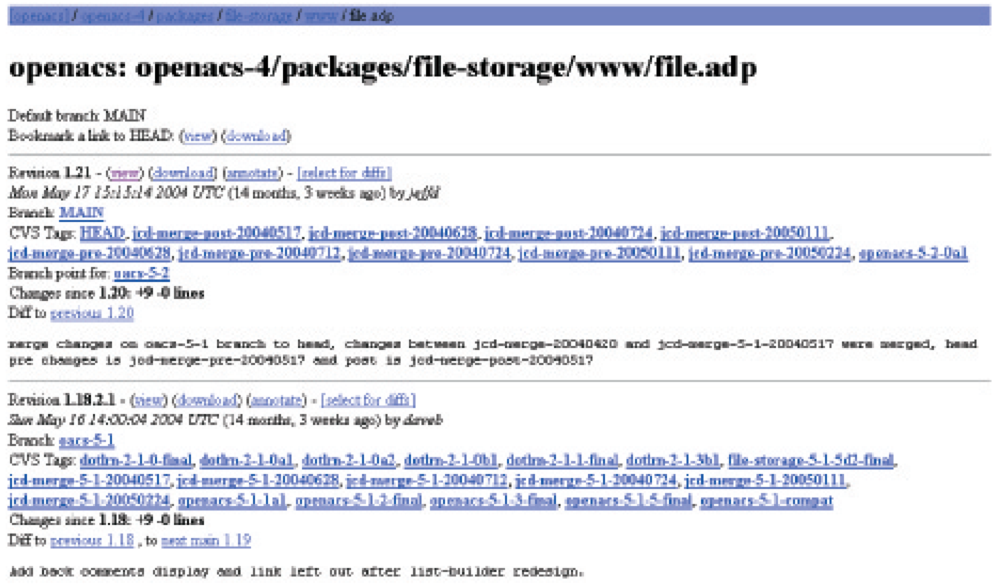
Patches

OpenACS Home : Bugtracker : OpenACS Bugs : Patches

Status (clear)	
Open	118
Accepted	366
Refused	163
Deleted	65
Apply to version	
4.6.1	1
5.0.0	8
5.1.0	1
5.1.2	1
HEAD	1
Component	
1-OpenACS General	45
2-dotLRN General	16
ACS Admin	10
ACS API Browser	12
ACS Authentication	7
ACS Bootstrap/Installer	5
ACS Content Repository	28
ACS Core Documents	4
ACS DateTime	2
ACS Developer Support	1
ACS Events	1
ACS Kernel	12
ACS Language	15
ACS Mail	1

Patch #	Summary	Status	Apply To	Component	Submitted
#695	patch	Accepted	*Undecided*	1-OpenACS General	6/26/2005
#685	Improved layout for admin/room-view	Accepted	*Undecided*	Room Reservation	6/07/2005
#684	Fixes input validation on the room and facilities admin forms	Accepted	*Undecided*	Room Reservation	6/07/2005
#683	Delete Reservations	Accepted	*Undecided*	Room Reservation	6/06/2005
#682	Improved Layout for some pages	Accepted	*Undecided*	Room Reservation	6/06/2005
#681	Automatic conversion of date format on room-reservations	Accepted	*Undecided*	Room Reservation	6/06/2005
#679	Backslashes in lt_Reservation_reservati	Accepted	*Undecided*	Room Reservation	6/06/2005
#678	Patch for datetime-procs-postgresql.xql	Accepted	*Undecided*	Room Reservation	6/06/2005
#669	This seems to fix it.	Accepted	*Undecided*	dotLRN Homework	5/27/2005
#667	Fixes the hex regexp in util_expand_entities_ie_style	Accepted	*Undecided*	ACS Tcl	5/12/2005
#654	fixes for cop-ui package on head to deal with ratings, clipboard, related-items	Accepted	*Undecided*	1-OpenACS General	3/17/2005
#653	fix for upload portrait	Accepted	*Undecided*	ACS Subsite	3/17/2005
#642	Permission check for calendar item delete	Accepted	*Undecided*	Calendar	2/25/2005
#641	de_DE catalog updates to match previous patch	Accepted	*Undecided*	ACS Authentication	2/20/2005

Figure 2.11: Screenshot of an OpenACS Patch Report



openacs / openacs-4 / packages / file-storage / www / file.adp

openacs: openacs-4/packages/file-storage/www/file.adp

Default branch: MAIN
Bookmark a link to HEAD: (view) (download)

Revision **1.21** - (view) (download) (annotate) - [select for diffs]
Mon May 17 15:15:14 2004 UTC (14 months, 3 weeks ago) by *jeffd*
Branch: MAIN
CVS Tags: HEAD, jcd-merge-post-20040517, jcd-merge-post-20040628, jcd-merge-post-20040724, jcd-merge-post-20050111, jcd-merge-pre-20040628, jcd-merge-pre-20040712, jcd-merge-pre-20040724, jcd-merge-pre-20050111, jcd-merge-pre-20050224, openacs-5-2-0a1
Branch point for: oacs-5-2
Changes since **1.20: +9 -0 lines**
Diff to previous 1.20

```
merge changes on oacs-5-1 branch to head, changes between jcd-merge-20040420 and jcd-merge-5-1-20040517 were merged, head
pre changes is jcd-merge-pre-20040517 and post is jcd-merge-post-20040517
```

Revision **1.18.2.1** - (view) (download) (annotate) - [select for diffs]
Sun May 16 14:00:04 2004 UTC (14 months, 3 weeks ago) by *daveb*
Branch: oacs-5-1
CVS Tags: dotlrn-2-1-0-final, dotlrn-2-1-0a1, dotlrn-2-1-0a2, dotlrn-2-1-0b1, dotlrn-2-1-1-final, dotlrn-2-1-3b1, file-storage-5-1-5d2-final, jcd-merge-5-1-20040517, jcd-merge-5-1-20040628, jcd-merge-5-1-20040712, jcd-merge-5-1-20040724, jcd-merge-5-1-20050111, jcd-merge-5-1-20050224, openacs-5-1-1a1, openacs-5-1-2-final, openacs-5-1-3-final, openacs-5-1-5-final, openacs-5-1-compat
Changes since **1.18: +9 -0 lines**
Diff to previous 1.18 , to next main 1.19

```
Add back comments display and link left out after list-builder redesign.
```

Figure 2.12: Screenshot of the OpenACS CVS Web Interface

Chapter 3

Dhruv: Supporting Bug Resolution in Open Source Software Communities

In this chapter, we describe the creation of the community Semantic Web prototype, Dhruv, motivated in the previous chapter. Dhruv performs two functions with respect to OSS bug resolution. First, it provides an *enhanced semantic interface* to messages posted during bug resolution. The enhanced interface allows community members to click on selected highlighted terms within the message, taking them to a *cross-links page*, which furnishes greater detail on the clicked term. The cross-links page primarily presents semantically related information about the term in the system and suggests related artifacts. Second, Dhruv provides a number of *message recommendations* of people, source code files, bug reports and discussions for each bug report message. These recommendations are determined by taking into account the semantic cross-links of each of the highlighted terms in the message.

In order to perform these two functions, Dhruv requires various components:

1. *Ontologies of the content, interactions and community*: To begin with, Dhruv requires an ontology that describes the structure of the project and provides a basis for determining how artifacts are related. We model all three layers of content, interactions and community of OpenACS, presented in Section 3.2. The content layer

is modeled in two ontologies, the code ontology (see Section 3.2.1) and the bugs ontology (see Section 3.2.2). These ontologies enable us to determine the location and context of given software objects and related bug reports. Essentially, these ontologies help identify the semantic context of a bug report message. The interactions ontology, presented in Section 3.2.3, describes the structure of interactions around bug reports, files and discussions in an OSS community. Using the interactions ontology, Dhruv can identify people who are experts in the area of the bug. Finally, the community ontology, presented in Section 3.2.4, describes the various roles in the community. Knowing the roles of people in the community, Dhruv can recommend people appropriate to their roles in the community. Thus, someone who has expertise in the area of the bug, but has never modified a file or submitted a patch, is likely to have the in-depth knowledge of the code required for fixing a bug.

2. *Metadata about various community artifacts*: Dhruv needs to identify meaningful terms and concepts within the community artifacts, namely the source code and the interaction messages of the community. There are two kinds of metadata: (a) references to code, files, packages, error traces, other bug reports and discussions, and (b) semantic concepts expressed as a technical vocabulary or jargon that is meaningful to community members. These extracted terms are highlighted by Dhruv within the message. In section 3.3, we discuss how these two types of metadata are automatically extracted and used to populate the ontologies created in section 3.2. In particular, we discuss the generation of metadata of type (a) from structured content in section 3.3.1 and metadata of type (b) from natural language text in Section 3.3.2. Finally, we discuss some of the issues that arise when attempting to generate metadata automatically for the OpenACS community in Section 3.3.3. The ontologies created in the previous step are used to provide the semantic context of the metadata of a bug report message.

3. *Heuristics to automatically link related objects and artifacts*: Having generated metadata for various kinds of software objects and community artifacts, Dhruv can now use the ontology relations created in the first step to identify the semantic context of the metadata generated in the previous step. This context consists of metadata

from the ontologies and related artifacts as determined by text similarity. We call the artifacts identified by the context of an extracted term the cross-links for the term. The cross-links for each extracted term in the message is presented in a cross-links page. The highlighted terms and cross-links pages together represent the enhanced semantic interface provided by Dhruv. We present various heuristics for generating cross-links for terms in Section 3.4. In particular, we discuss the heuristics Dhruv uses to identify artifacts and objects from noun phrases (Section 3.4.1), from code terms (Section 3.4.2) and from references to other artifacts (Section 3.4.3).

4. *Recommendation procedures to suggest related artifacts during bug resolution*: Given the semantic context or cross-links for message terms, Dhruv can now generate message recommendations. Dhruv gathers all the cross-links for a message and then prunes and ranks the list of artifacts using a number of heuristics. In Section 3.5, we discuss the heuristics for generating recommendations from the cross-links created in Section 3.4.

3.1 Dhruv: A Prototype Semantic Web to Support OSS Bug Resolution

The core idea of Dhruv is to realize a prototype Semantic Web to support bug resolution in the OpenACS community. Dhruv supports bug resolution in two ways. Dhruv captures metadata about various artifacts created as part of the development history of OpenACS. It then presents related metadata to the developers in a natural way as they attempt to fix bugs. In addition, Dhruv suggests artifacts that are likely to be relevant to a developer working on a community activity, such as resolving OpenACS bug reports.

As discussed in the previous chapter, Chapter 2, Dhruv essentially forms a knowledge base with concept relations and instances. It uses its knowledge of OpenACS artifacts to form an OpenACS project memory that links related artifacts. OpenACS artifacts include the source code and documentation, discussion forum messages, bug and patch reports, and chat logs.

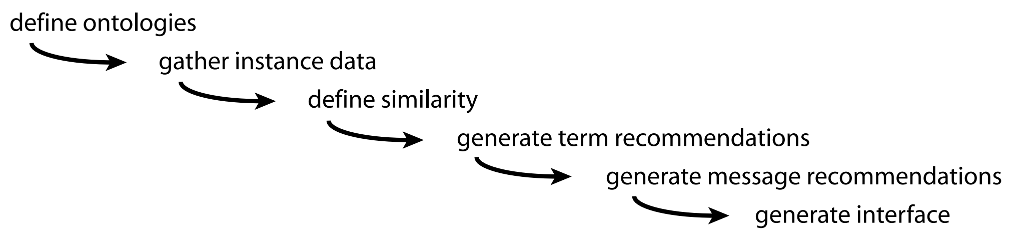


Figure 3.1: Dhruv Creation and Function

The creation and function of Dhruv, outlined in Figure 3.1, will be described in the rest of the chapter. The first step in creating Dhruv involves creating the ontologies that are part of Dhruv's knowledge base by examining the artifact sources that the ontologies model. The next stage involves gathering instance data to populate the Dhruv ontology concepts and roles. Having created a knowledge base with an ontology and metadata, the following stage creates links between instances that are semantically related. Dhruv uses the metadata and links to present useful background information to an OpenACS community member in a natural way. In addition, Dhruv uses the links to make recommendations for related artifacts.

3.2 The OpenACS Community Ontology

The OpenACS Community Ontology used by Dhruv has been designed to support a developer attempting to resolve a bug. For a given bug report, the developer must be able to query Dhruv for related bug reports and forum discussions. In addition, Dhruv should be able to indicate (roughly) the area of code where the bug is likely to be located and other developers who have expertise in that area of code. Such developers are likely to be able to help resolve the bug.

The top-level OpenACS ontology essentially consists of four sub-ontologies, shown in Figure 3.2. The purpose of the ontologies is to model the structures of the three layers of the OpenACS community, enabling artifacts and information from various parts of the project to be linked. The content layer of the OpenACS Community is modeled through the code ontology, which describes the project source code, and the bugs ontology, which describes submitted bug and patch reports and their attributes. These two ontologies are connected, because code files modeled in the code ontology contain bugs that are modeled in the bugs ontology. The interactions layer of the OpenACS Community is modeled by the interactions ontology, which covers the various kinds of interactions and discussions that take place on the project website. These often refer to the bug reports and source code of the OpenACS Community. Finally, the community ontology describes the people in the OpenACS Community and various roles they assume within the community. These roles are determined through their interactions within the community and consequently, the community ontology builds on the interactions ontology.

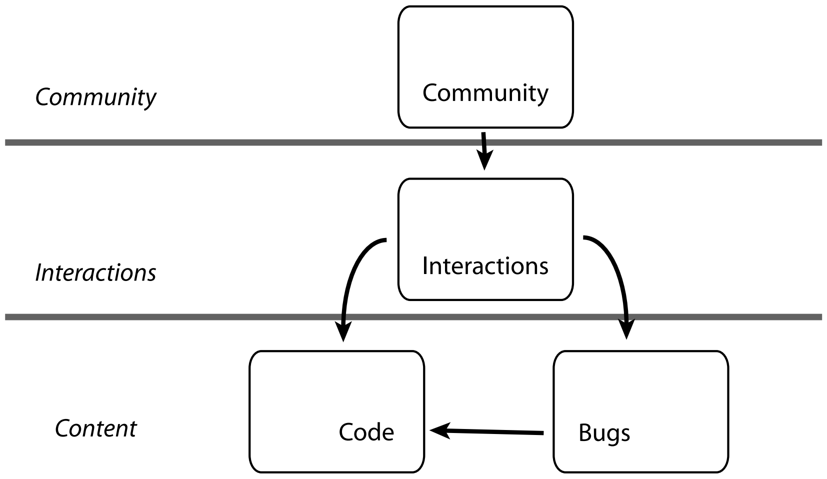


Figure 3.2: An overview of ontology hierarchy

The OpenACS Community ontology has been segmented into the four sub-ontologies for various reasons. The sub-ontologies reflect the three-layered online professional community framework of content, interactions and community. The content layer ontology has been further sub-divided into the code and bug ontologies, since they are sufficiently different domains, each meriting its own ontology for clarity. Each of the ontologies are relatively modular and are consequently usable for the most part independently of each other. The Semantic Web is intended to accommodate multiple ontologies. Segmenting the OpenACS Community ontology allows us to explore the use of multiple ontologies for a single setting, albeit non-rigorously.

Due to the multiple ontologies in use for the OpenACS community, in the following we use the convention `ont:concept` to refer to a class `concept` defined in an ontology `ont`.

We construct the artifact ontology in OWL (Web Ontology Language) DL, so named due to its correspondence with *description logic*, a decidable fragment of first-order logic that possesses desirable computational properties for reasoning[1]. However, unlike OWL, we make the *unique names assumption*, that two different names always refer to two different individuals. Although this assumption may not hold in general on the Web, it simplifies reasoning considerably and most DL reasoners, including RACER, make the same assumption by default. We therefore only need to explicitly assert that two individuals are equivalent, a case that occurs relatively infrequently in our context.

3.2.1 The Code Ontology

The OpenACS ontology is intended to support for bug resolution, but its core models the software source code and its evolution. The source code is the most important artifact in understanding and resolving bugs. All bugs occur somewhere in the source code and the ensuing discussion around the bug also takes place in referential context of the source code. The software source code ontology is therefore the most critical portion of the ontology, since it models the source code, which is key to interpreting the semantics of other artifacts, especially message-based artifacts.

The OpenACS architecture and organization must be understood in order to design the software code ontology appropriately. In the next section, we give a brief overview of the OpenACS architecture and then proceed to describe the OpenACS code ontology.

[1]Unlike database reasoning, OWL makes an open world assumption, i.e. things left unspecified are assumed to be unknown.

OpenACS Architecture and Organization

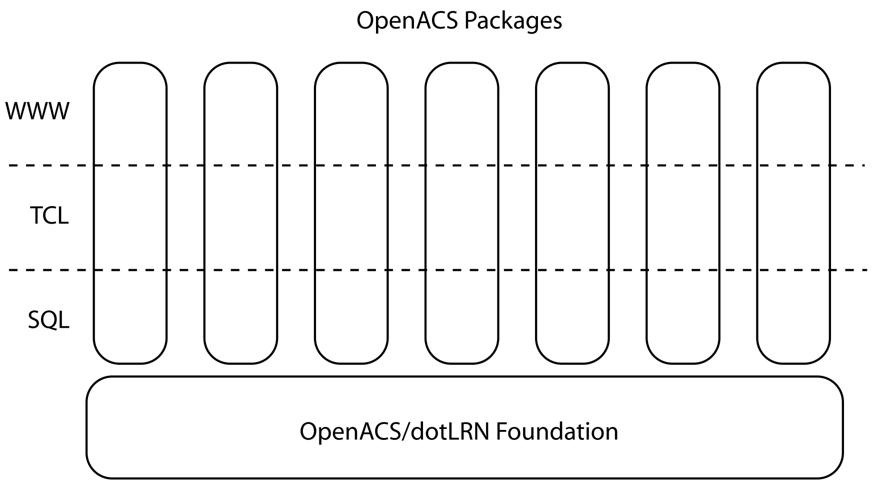


Figure 3.3: A high-level view of the OpenACS software architecture

The OpenACS software is organized as a toolkit, from which a number of tools can be chosen and put together to realize a Web community application. A very high-level view of its architecture is shown in Fig. 3.3. There are numerous packages which build on a common foundation. Each of the packages provides a functionality or tool, such as Web forums, shared file repository, a web log and so on. There are around 170 maintained packages in the OpenACS/dotLRN software platform. In the following, we briefly discuss the internal architecture of an OpenACS package and its organization of files. This will turn out to be important when we consider the file recommendations made by Dhruv.

Each OpenACS package is internally organized according to the model-view-controller (MVC) architecture. This is also reflected in the organization of the source code. Each package has three sub-directories: `sql` to store the code that implements the package model, `tcl` to store the code (primarily in the programming language TCL) that implements the controller and `www` to store the code that implements the view or graphical interface of the application.

Within each package, the model is implemented through database manipulation code, typically stored in SQL files. OpenACS maintains separate database scripts for two popular relational databases: Oracle, a commercial product and PostgreSQL, another open source software product. Scripts that implement the same functionality, but are meant for different databases are differentiated by directory. Thus, the `sql` directory has two subdirectories, `oracle` and `postgresql`, which contain source code for the corresponding databases.

The `tcl` directory of a package contains the code to implement the package controller. Each TCL file in the package can have a corresponding XQL file, which stores in XML Query Language format, the database queries made by functions in the TCL file. For example, the following are a set of related files found in the `tcl` directory of the `forums` package.

```
message-procs.tcl
message-procs.xql
message-procs-oracle.xql
message-procs-postgresql.xql
```

The file `message-procs.tcl` contains the TCL code to implement forum message functionality, such as creating and editing a forum message. For this, the TCL functions need to access the database to create and update the messages table. Database queries, whose syntax is independent of database are stored in `message-procs.xql` and database-dependent queries are stored in the files `message-procs-oracle.xql` and `message-procs-postgresql.xql` respectively.

The Web interface of each package is implemented by code stored in the `www` directory. OpenACS applications are typically database-driven websites. Therefore, the web interface of individual packages is implemented through HTML pages that are dynamically generated based on a Web pages template and the contents of an underlying database. The Web page templates are written in ADP (AOLserver Dynamic Page) and the code that dynamically populates them is written in TCL. The file organization once again reflects this connection, so the following two files in the `www` directory of the `forums` package are related.

```
message-view.adp
message-view.tcl
```

The file `message-view.adp` contains the ADP template for viewing forum messages and the file `message-view.tcl` contains the TCL code responsible for querying the database (through functions in the `tcl` directory) and passing the results onto the ADP file.

With an understanding of the OpenACS architecture, we can now describe the code ontology. The OpenACS code ontology has about 24 concepts and 19 properties. The subclass hierarchy of concepts in the code ontology is shown in Fig. 3.4. The root concept, `owl:Thing`, is a top-level class, containing all instances in the knowledge base. A direct subclass of `owl:Thing` is the concept `community:Resource`, which represents all artifacts and people in the community and will be elucidated further in section 3.2.4. Here, we discuss a direct subclass of `community:Resource`, namely `code:SoftwareObject`, which represent the top-level class for all software objects in the OpenACS code base.

In the following, we discuss each of the subclasses of `code:SoftwareObject` in detail.

Package

OpenACS packages are modeled through the `code:OACSPackage` class, which is a subclass of the concept `code:Package`. Instances of `code:OACSPackage` represent a particular tool in the OpenACS toolkit, such as `forums`, `calendar` or `photo-album`. Another subclass of `code:Package` is `code:OraclePackage`, which represents a 'package' object in the Oracle database and contains a set of Oracle function definitions. There is no corresponding object in PostgreSQL.

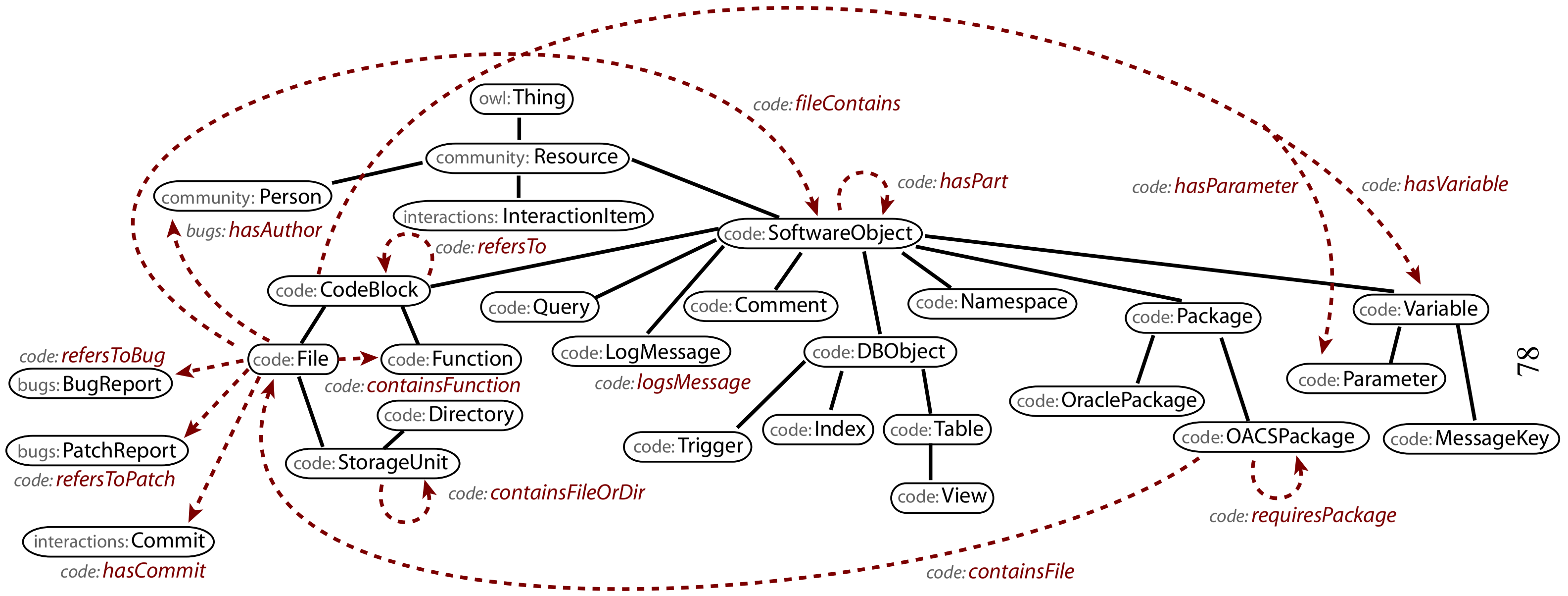


Figure 3.4: The Dhruv Code ontology

Code Block

The concept code:CodeBlock is a container class, describing software objects containing a set of actions to be executed. In OpenACS, there are primarily two kinds of such objects, functions (modeled by code:Function) and files (modeled by code:File). In OpenACS, functions are written in either TCL or SQL. TCL functions drive the website processing and page display, while SQL functions are for specialized database functionality. In addition to functions, OpenACS also uses files as containers for actions. For example, the ADP files.

Namespace

OpenACS functions are organized into namespaces, which are represented in the code ontology by the concept code:Namespace. Namespaces can be nested. By knowing in which namespace a function belongs, we can find related functions by looking at other functions in the namespace. The namespace also provides a higher-level description.

Variables

Variables within functions are modeled through the concept code:Variable, which has two specializations code:Parameter and code:MessageKey. The concept code:Parameter represents variables whose scope is local to a function. Message keys, on the other hand, are special global variables, representing package parameters. They are often used to implement localization.

Database Objects

The concept code:DBObject models OpenACS software objects that are associated with a database. Thus, subclasses of code:DBObject include the class code:Table, which models a relational table in OpenACS. code:OraclePackage, discussed above, is also a database object and is therefore a subclass of code:DBObject. The other two subclasses of `code:DBObject` are `code:Index` and `code:Trigger`, which represent database indices and triggers, respectively.

Query

The concept `code:Query` models database queries made by TCL functions. If the TCL functions are located in a file `file.tcl`, then the queries are always located in a file `file.xql`. Instances of class `code:Query` are linked to instances of `code:Function` through two properties: the object property `code:invokedBy` and its inverse property `code:invokesQuery`.

Comment

The concept `code:Comment` models software code comments within a source code file. An instance of `code:Comment` may be a documentation comment for a function or namespace, such as

```
This procedure, given a content item and a privilege,
checks to see if there are any children of the item on
which the user does not have that privilege.  It returns
0 if there is any child item on which the user does not
have the privilege.  It returns 1 if the user has the
privilege on every child item.
```

or an inline comment such as

```
# FIXME: db_blob_get_file is failing when i use bind
variables
```

`code:Comment` also has a functional datatype property `hasCommentText`, which links an instance of `code:Comment` to the textual string of the comment.

Log Messages

In analogy to the concept `code:Comment`, the concept `code:LogMessage` models logging messages within a source codefile. During the running of the OpenACS system, these logging messages are written to a log file and are typically used by developers and users to understand the internal state of the OpenACS system. This is usually helpful in diagnosing why and how the system reached an erroneous state. Community members encourage people filing bug reports, where the system returns an error, to include relevant portions of the log file in the bug report. In the line of code below, the first word `ns_log` is a keyword indicating that a log message should be printed of type `Warning` with the message `Multiple ... script`.

```
ns_log Warning "Multiple definitions of ad_page_contract
filter "$name" in $script and $prior_script"
```

We model the textual component of the message as a string, which is linked to by an instance of `code:LogMessage` through the functional datatype property `code:hasLogMessageText`. The types of log messages are not modeled in our ontology, since these primarily signal the severity of a problem to other developers. Log message types are not currently used by Dhruv.

In addition to these software objects, the code ontology also contains the concept `code:StorageUnit`, which refers to the files and directories where particular blocks of code are stored. Individual commits made by people to files are modeled through the concept `code:Commit`.

Several kinds of reasoning capabilities were envisioned for the code ontology, which influence its design. These are:

- *Identifying concepts that appear in bug reports.* Certain concepts, such as variable names, function names and log messages, are likely to appear in bug reports and provide valuable hints about the code that is involved in producing the bug. These concepts should be modeled by the ontology in order to link from information in a bug report to areas in the code that cause the bug. The code ontology therefore contains concepts, such as `code:LogMessage` and `code:Comment`, and properties, such as `code:hasLabel` and `code:hasCommentText`.

- *Tracing invocation dependencies for code.* The run-time behavior of code depends on its control flow structure and invocations of other functions and procedures. Although the behavior of a procedure may depend on the context within which it is invoked or on the procedures it invokes in turn, these dependencies are difficult to trace for someone unfamiliar with the code. The code ontology therefore models the procedure invocations within the code, both to present a localized view of code connected by control flow and to identify which areas of code are affected by a particular bug. This is achieved by the property `code:invokes` and its two subproperties `code:invokesFunction` and `code:invokesQuery`.

- *Identifying people who committed changes to code.* The evolution of a code base is logged in CVS commit logs as commits made by developers to various files. It is important to model commits in our ontology to be able to locate people who have often committed to a particular file and are likely to have expertise in the code contained in the file. Consequently, our ontology contains a concept `code:Commit`, which has the following properties: `code:hasCommit`, `code:hasCommitter`, `code:committedOnDate` and `code:hasCommitText`, which points to the log message associated with the commit. In case the commit was made in response to a bug or a patch report, this is captured through the property `code:refersToBug` or `code:refersToPatch`.

In order to support the last reasoning task above, we also model *the storage structure of the code*. Code packages and procedures are usually distributed across several files. In order to connect commits performed on files, to actual blocks of code, the ontology models storage structure of the code, i.e. the file(s) in which a procedure (or package) is located. This is achieved through the concepts of `code:File` and `code:Directory` and the properties `code:fileContains` and `code:hasFileOrDir`.

Many of the concepts, such as `code:Function` and `code:Table`, do not model the real-world objects, such as functions and tables, completely. For example, they do not model columns of the table and return values of functions. However, this is inconsequential, since comprehensive modeling is not required by any of the reasoning tasks and is in fact detrimental to efficient use of the concepts. The purpose of the code ontology is therefore primarily to model the location, type and partof relations for significant code objects.

3.2.2 The Bug Ontology

The bugs ontology is the other portion of modeling the content layer. The bugs ontology models the information in the OpenACS Bugtracker: the bug reports, their attributes and the discussions around them. The bugs ontology is much simpler than the code ontology, containing 11 concepts and 15 properties, as shown in Fig.3.5. The classes and properties in the ontology are explained below in detail:

Bug and Patch Reports

The core classes in the bugs ontology are `bugs:Report` and its two direct subclasses are `bugs:BugReport` and `bugs:PatchReport`. These represent a bug report and a patch report respectively, filed in the OpenACS Bugtracker.

Report Attributes

Each report can have several attributes, describing various characteristics of the report. Report attributes are modeled through the concept `bugs:ReportAttributes`. The OpenACS BugTracker makes use primarily of three attributes priority, resolution and severity. These are modeled as classes `bugs:Priority`, `bugs:Resolution` and `bugs:Severity` and are linked to the report through the functional object properties `bugs:hasPriority`, `bugs:hasResolution` and `hasSeverity` respectively.

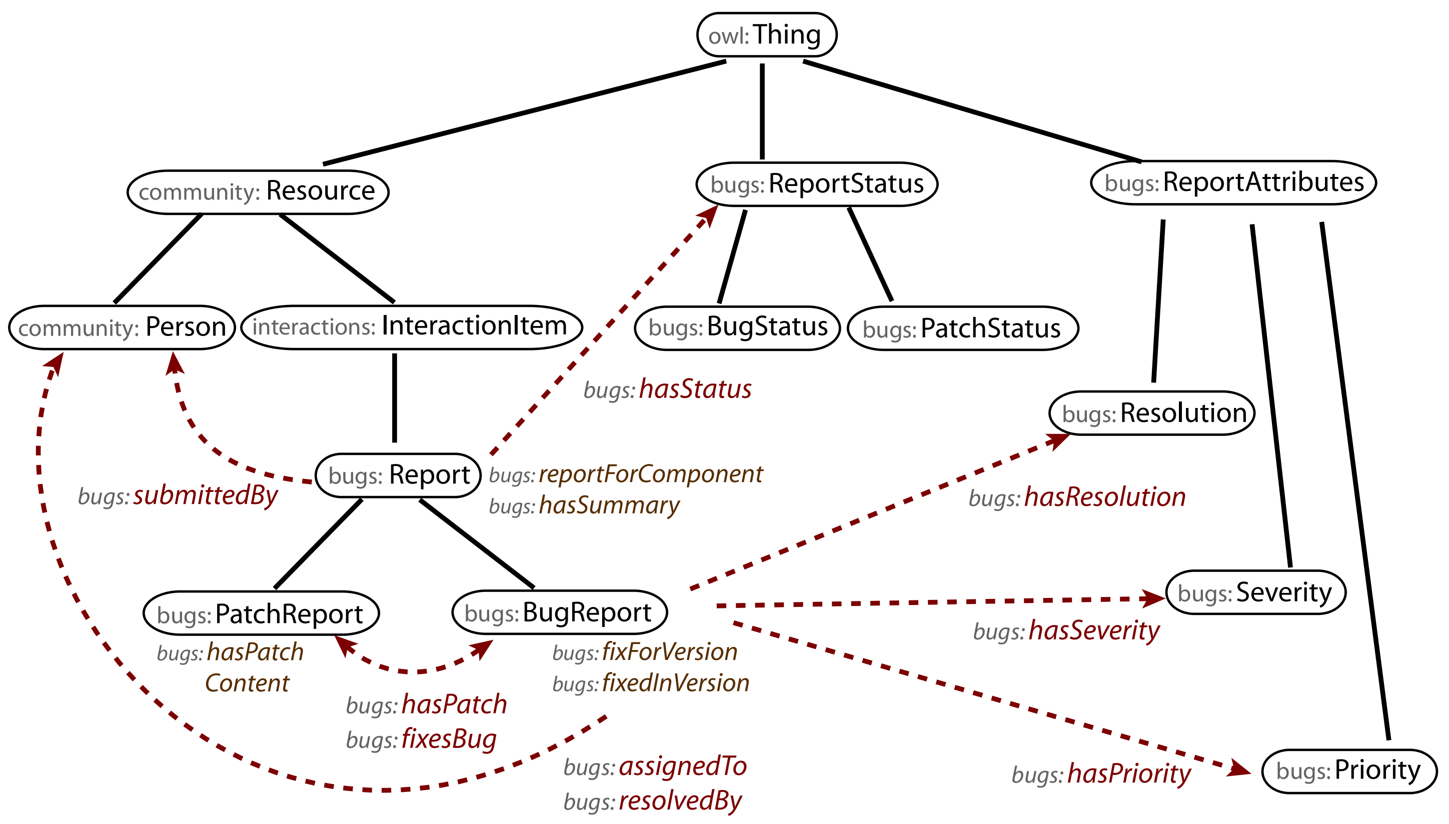


Figure 3.5: The Dhruv Bugs Ontology

There are three different levels of priority for each bug report, which represents the urgency with which the bug should be resolved. These levels are modeled as instances of the class `bugs:Priority`. The base level is `bugs:Normal`, which denotes normal priority for the bug. A bug report that is more serious, may be designated to be fixed before the next release of the software, i.e. it would have priority `bugs:MustFix`. The highest level of priority is denoted by the level `bugs:Urgent` for bugs which must be fixed immediately, because they are blocking other work on the system.

The severity of each bug report represents the criticality of the bug. Like the priority of a bug, the severity of a bug is modeled by the class `bugs:Severity`, with the six levels of severity modeled as instances of this class. A bug may be simply an issue of aesthetics, in which case, it is noted as having severity `bugs:Cosmetic`. A bug may otherwise be `bugs:Trivial` to resolve or an `bugs:Inconvenience` if a workaround exists that addresses the bug. If a core functionality of a package is broken, then the bug is marked as `bugs:FunctionBroken`. If a core functionality of the OpenACS system is broken or the system crashes and for bugs involving data loss or security, the bug is marked as being `bugs:Critical`. If nothing is broken, the bug may be marked as merely an `bugs:Enhancement`.

Report Status

Most of the above attributes of a bug report also carry over to a patch posted to the bug report. However, the status of bug reports, modeled by `bugs:BugStatus`, and patch reports, modeled by `bugs:PatchStatus`, are markedly different and are discussed below. They are linked to the bug or patch report through the functional object property `bugs:hasStatus`.

Unresolved bug reports are in the state `bugs:Open`. If a solution has been found for the bug, the bug report is marked as `bugs:Resolved` and once the solution has been verified by the person who reported the bug, the bug report is marked as `bugs:Closed`. People are encouraged to state the reason why a bug report is considered resolved or closed. If the bug was fixed, the bug report may then be noted as being either `bugs:Closed_Fixed` or `bugs:Resolved_Fixed`. If the bug report is deemed to report the same bug as another bug report, it will be noted as being an instance of one of the classes `bugs:Closed_Duplicate` or `bugs:Resolved_Duplicate`. If it has been decided that the bug report does not really represent a bug and is the way the software is expected to function, the bug report will be marked as either `bugs:Closed_ByDesign` or `bugs:Resolved_ByDesign`.

Finally, bug reports may not be reproducible in a test environment (in which case, they are marked as `bugs:Closed_NotReproducable` or as `bugs:Resolved_NotReproducable`) or their resolution may simply be postponed to a later date (`bugs:Closed_Postponed` or `bugs:Resolved_Postponed`).

Patch reports are much simpler and may be in only four different states. They are either `bugs:Open`, `bugs:Deleted`, `bugs:Accepted` or `bugs:Refused`. A patch may be deleted, for example, because a more comprehensive patch exists elsewhere. Patches can also be refused, if they do not resolve the bug or cause other problems to arise. If the patch is appropriate and fixes the bug, it is accepted.

Various attributes of reports have been modeled by subclasses of the class `bugs:ReportAttributes`. These attributes are captured mainly to be able to define rules over the ontologies some time in the future. An example of such a rule might be 'For a `bugs:BugReport` with `bugs:urgent` `bugs:Priority` in `bugs:Component` `ACS-Authentication`, always mark `John` as relevant.'

As with the code ontology, there are a few reasoning tasks that have been envisioned for the bugs ontology. It is often helpful for a developer to know who committed changes in the near past. Some of these changes are likely to have introduced a bug. In addition, the developer may also want to know which developers are likely to have expertise in the area of the bug. Both of these questions can be answered by examining the commit logs for the code that produces the bug. While it can be very difficult to identify the relevant people exactly, the Dhruv knowledge base can determine the people who made changes to code referred to in the bug. Using the code ontology, Dhruv can link bugs to units of code by identifying code concepts that appear in the bug report. Dhruv can then identify the people who committed to the files that contain the units of code.

3.2.3 The Interactions Ontology

On top of the code ontology and the bugs ontology, lies the interactions ontology, which models community interactions around artifacts. The interactions ontology therefore models people's actions on bug reports and on discussion forums. This ontology captures the semantics of the interactions layer and is shown in Figure 3.6.

In the interactions ontology, interactions are modeled as follows: individual artifacts or interaction items may have `interaction:Message`s made by instances of the class `community:Person`. Each instance of `interactions:Message` is differentiated into several types, such as `interactions:OpenMessage`, which refers to the action performed on the artifact by posting the message. These concepts are discussed in detail below.

Interaction Items

An interaction item represents an artifact, an instance of `community:Resource`, around which an interaction may take place. The class `interactions:InteractionItem` has three direct subclasses `interactions:DiscThread`, `bugs:Report` and the class `code:File`. The classes `bugs:Report` and `code:File` have been discussed previously and support interactions in the form of discussions around bug reports and commit sequences on files respectively. The class `interactions:DiscThread` represents the set of all discussion threads in the OpenACS Forums.

Each instance of the class `interactions:InteractionItem` is linked to the interactions around it through the functional object property `interactions:hasMessageSequence`. The interactions are represented through the class `interactions:MessageSequence`, which is discussed next.

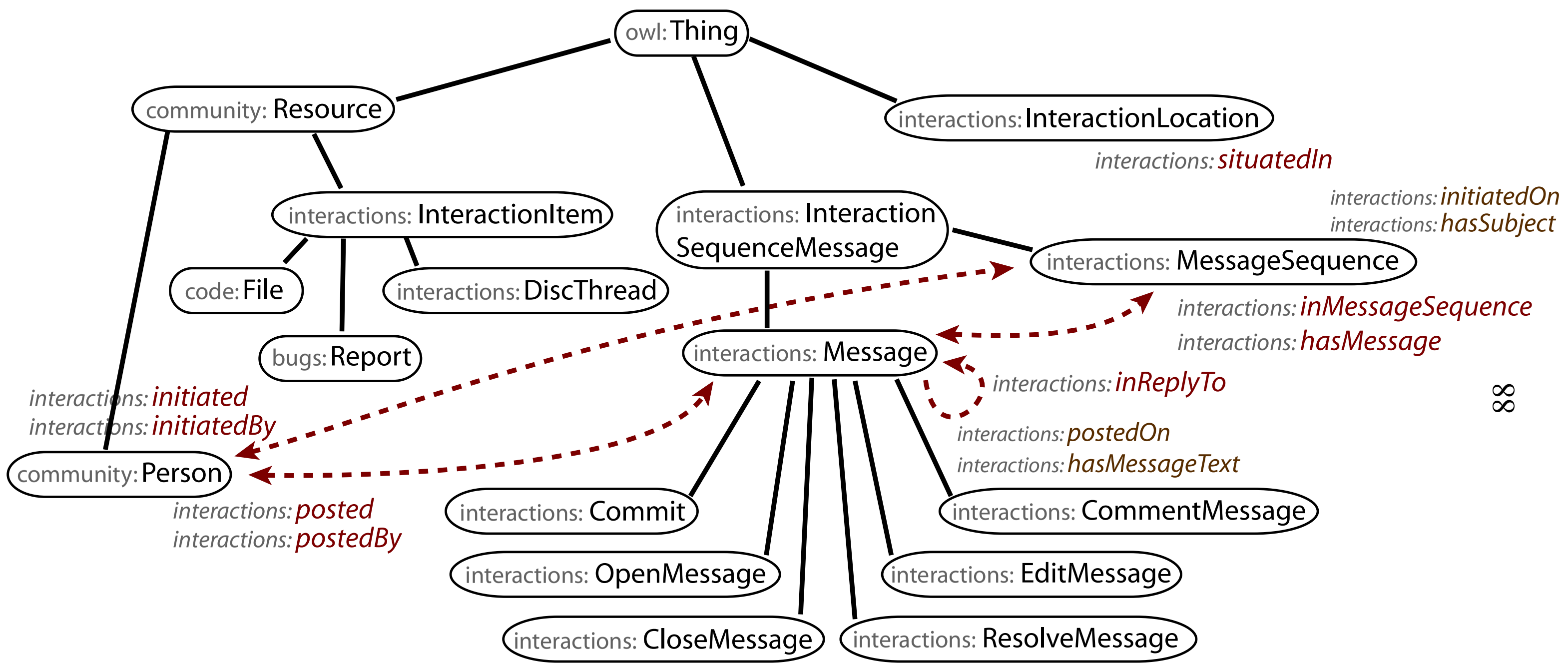


Figure 3.6: The Dhruv Interactions Ontology

Messages and Message Sequences

The classes `interactions:Message` and `interactions:MessageSequence` are represent individual messages and sequences of messages respectively. The class `interactions:Message` has several object and datatype properties describing the various attributes of a message. A message is `interactions:postedBy` a person and `interactions:postedOn` a particular date. The message has a text string as subject, linked to by `sinteractions:hasSubject`. The message might have started a new discussion or it may be `interactions:inReplyTo` another instance of the class `interactions:Message` In the latter case, both messages are linked by the object property `interactions:inMessageSequence` to the same instance of the class `interactions:MessageSequence`. Each message also has an associated message text, of type string, linked to by the functional datatype property `interactions:hasMessageText`.

Message Types

There are several kinds of messages in OpenACS discussions, which are modeled as subclasses of `interactions:Message`. By representing the various classes of messages in the interactions ontology, we are able to classify people into various kinds.

Messages can be of six types:

- `interactions:OpenMessage` is a message that initiates a discussion. It is the first message in any interaction sequence and has no message that it is in reply to. This message may be posted to an instance of `bugs:Report` or the class `interactions:DiscThread`.
- `interactions:CommentMessage` is a message that is in reply to another message on a `bugs:Report` or a `interactions:DiscThread`.
- `interactions:EditMessage` is a message that is posted only to `bugs:Report`. It is used to edit some of the attributes of the bug or patch report.

- `interactions:ReassignMessage` is a message posted to a `bugs:BugReport`, indicating that the resolution of the bug report has been assigned to a new person.

- `interactions:ReopenMessage` is a message posted to a `bugs:BugReport`, indicating that a bug report previously thought to have been resolved or even closed has been reopened, because the bug still persists.

- `interactions:ResolveMessage` is a message that is only posted to an instance of the class `bugs:BugReport` and is used to indicate that the bug in question is considered to be fixed.

- `interactions:CloseMessage` is a message that usually follows a message of type `interactions:ResolveMessage` and indicates that the bug report is formally closed. This is usually done by the person who sent the `OpenMessage`, i.e. the person who first noticed the problem, after verifying that the bug is indeed fixed.

- `code:Commit` is a message that is posted to a `code:File` as part of a commit log.

Bug report comments are modeled because they contain valuable information that elaborates on the bug, its symptoms, possible fixes and trade-offs. Such information is very useful, for example, to a developer wishing to fix a similar bug. In addition, bug comments indicate the people who participated in the resolution of the bug and are therefore likely to have some expertise in the area of the bug.

Similarly, comments or messages posted on web-based discussion forums are also modeled, since they may contain information that a developer needs to be aware of while fixing bugs.

3.2.4 The Community Ontology

On top of the interactions ontology, lies the community ontology, which models the online community in terms of the people taking part in the community interactions around artifacts and their roles. The community ontology therefore captures the semantics of the community layer and is shown in Figure 3.7.

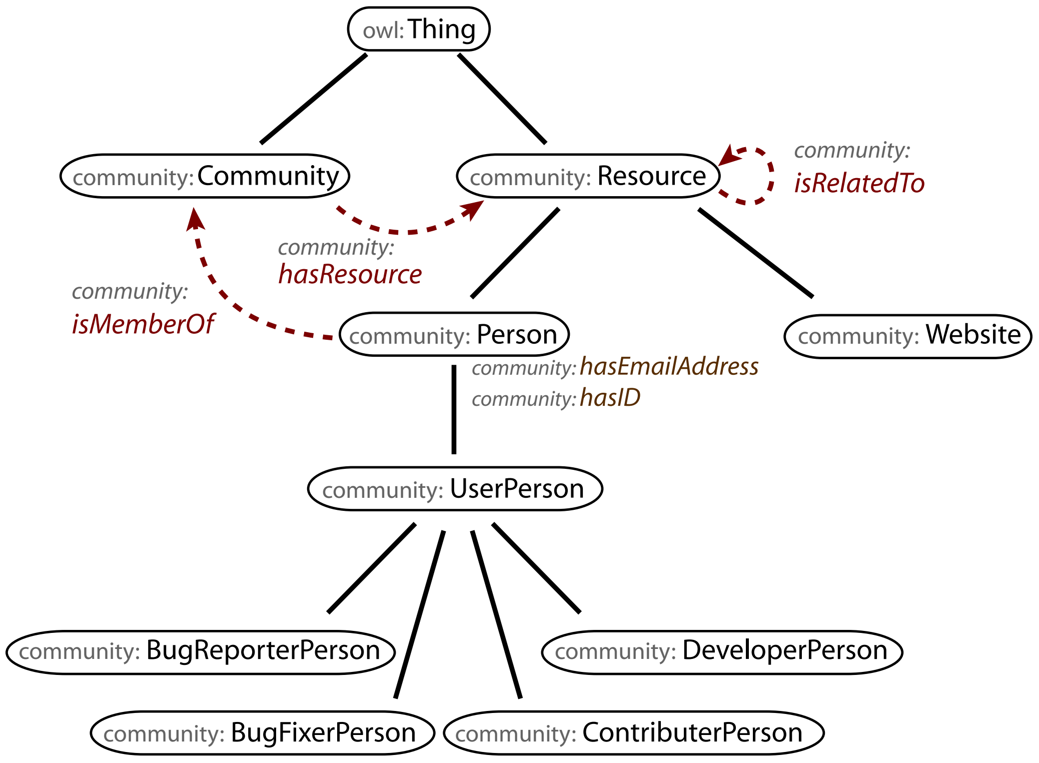


Figure 3.7: The Dhruv Community Ontology

There are three key classes in the community ontology. Firstly, the class `community:Community`, which contains numerous `community:Resources` and a number of instances of class `community:Person`.

Community

The class `community:Community` represents an online community. An instance of `community:Community` is linked to several shared resources through the object property `community:hasResource`.

Resources

The concept `community:Resource` represents all resources that can be used for support or help by the community. These are primarily community artifacts, such as source code files, bug reports, discussion threads and people. Resources could potentially include external websites and discussions, but they are out of the scope of this work. The class `community:Resource` has four subclasses: `community:Person`, which represents a community member; `code:SoftwareObject`, which represents all OpenACS objects and `interactions:InteractionItem`, which represents all artifacts with interactions around them, i.e. bug reports, discussion threads, source code files etc.

People

The concept `community:Person` represents a member of an online community, linked to an instance of `community:Community` through the object property `isMemberOf`. A `community:Person` can have multiple associated email-addresses and multiple community identifiers. These are represented by the two following datatype properties: `community:hasEmailAddress` and `community:hasID` respectively.

The community ontology is also linked to the FOAF ontology [BM04]. In particular, every instance of `community:Person` is also a `foaf:Person`. At the moment, this subclass relationship is not utilized. However, tools built for FOAF can also make use of the OpenACS `community:Person` information. As more ontologies are defined within the Semantic Web, the OpenACS ontologies can be linked with additional external ontologies, increasing the usefulness of the OpenACS ontologies.

A community member can have several roles in the community, especially in an open source software community. A person could be a user of the software, an occasional contributor of code or a core developer. These roles of community members are represented by subclasses of `community:Person`. They are:

- `community:UserPerson` is a person who has used or attempted to use the software and has had some interaction with the community. Thus, anyone who has posted a message in the community is classified as a user of the software. Note that this does not include users who might lurk around without participating in community interactions. They are anyway invisible to the community and cannot function as a resource.

- `community:BugReporterPerson` is a person who has filed at least one bug report and thus initiated a discussion around the bug. Bug reporters are users who have a higher level of commitment to the software, to the extent of actively participating in improving its quality.

- `community:BugFixerPerson` is a person who has resolved or fixed at least one bug. Bug fixers have demonstrated commitment and some level of competence in the software by fixing existing bugs.

- `community:ContributerPerson` is a person who has submitted at least one patch report. A patch report may be filed to fix an existing bug, in which case the contributor is essentially a bug fixer. However, patch reports are also used by peripheral community members to contribute code and enhancements to the community. Thus, the category of contributor encompasses bug fixers, but goes beyond them.

- `community:DeveloperPerson` is a person who is a core community member and has commit privileges in the community. Thus, an instance of the class `community:DeveloperPerson` is someone who has committed at least once to the shared code base. Developers are often also responsible for the technical direction of the community.

3.3 Generating Metadata

Having defined classes and roles in the OpenACS ontology, namely the TBox, we can now consider how to define the instances that populate these classes, namely the ABox. These instances are gathered from the artifacts, namely source code files, CVS commit logs, bug and patch reports and discussion threads.

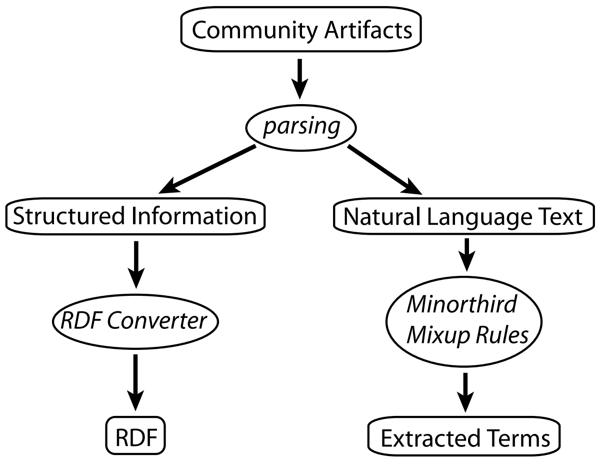


Figure 3.8: An overview of the metadata generation process

A schematic of the metadata generation process is shown in Fig. 3.8. The metadata generation process uses a combination of hand-written parsing rules and information extraction patterns. We attempted to generate the instance data automatically as far as possible. There are several reasons for this. Firstly, it would be infeasible to define all the instances in the ontology manually. This is true for any ontology that aims to capture any sizable domain. Although it requires some upfront effort to teach a system like Dhruv to acquire data automatically, once the system is functioning, the incremental effort of data generation on the ontology developers and users is negligible.

By restricting ourselves to automatic instance acquisition, we also limit ourselves to capturing information, that is high in structural semantics, but may be relatively shallow in its interpretational semantics. For the purposes of this exploratory study, though, this restriction is not severe. Important instances not captured automatically can always be added manually to the ABox. Naturally, the decision to acquire instances automatically has influenced the design of the TBox to an extent, in that only those concepts and properties were retained, whose instances could be determined automatically.

The artifacts used in the metadata generation process are:

- Software source code from OpenACS packages. We only used considered TCL, SQL, XQL and ADP files. HTML files and additional glue code in OpenACS was not taken into account.
- CVS commit logs for the OpenACS packages' source code.
- Bug reports submitted in the OpenACS Bugtracker.
- Patch reports submitted to the OpenACS Bugtracker.
- Discussion threads in the OpenACS Web Forums.

3.3.1 Generating Metadata from Structured Data

These artifacts were input to a parser, which extracts structured information, that is to be captured for the content and interactions layer ontologies. For example, the fragment of source code from the file `forum-procs.tcl` below

```
ad_proc -public forum::list_forums {
    {-package_id:required}
} {
    List all forums in a package
} {
    return [db_list_of_ns_sets select_forums {}]
}
```

is parsed to extract the following information:

```
code:Namespace forum
code:Function list_forums
code:containsFunction forum list_forums
code:Parameter package_id
code:hasParameter list_forum package_id
code:Comment "List all forums in a package"
code:hasComment list_forums "List all forums in a package"
code:Function db_list_of_ns_sets
code:refersTo list_forums db_list_of_ns_sets
code:fileContainsDefOf forums/tcl/forums-procs.tcl
forum::list_forums
```

The parser is written in Perl and uses regular-expression pattern matching to identify functions, namespaces, queries, comments etc. In the example above, the keyword `ad_proc` can be used to identify a function definition and other features of the syntax used to identify the function name, function comment and so on. Of course, this process is not entirely error-free. To do this really thoroughly, Dhruv would need to be able to understand the syntax of TCL, ADP, SQL and XQL, to parse the programs. It would still not be entirely successful, because OpenACS has defined its own object system on top of TCL. For the time being, we therefore wrote hand-coded rules to gather the relevant information. The parser also generates unique identifiers for each of the instances. For simplicity, the unique identifiers are not shown in the example above. The extracted information is then transformed to RDF using a RDF converter. The RDF converter, also in Perl, cleans the extracted information for RDF.

3.3.2 Generating Metadata from Natural Language Text

In addition, the parser also extracts natural language text from the artifacts. This is primarily the text of discussions, bug report and patch report messages, CVS commit logs and comments in the source code. This text is processed using Minorthird Mixup rules to extract the following:

- Bug report and patch report references. These were captured when they appeared as the following

```
bug 23
bug #1245
bugs #1245 and #1243
http://openacs.org/bugtracker/openacs/bug?bug_number=1445
patch 342
patches 343 and #342
http://openacs.org/bugtracker/openacs/patch?patch_number=342
```

- Discussion thread references. These were captured when they appeared as the following

```
http://openacs.org/forums/message-view?message_id=131005
```

- Filenames and file paths. These were captured when they appeared as the following

```
forums/tcl/forum-procs.tcl
message-procs.xql
/packages/acs-mail/www/doc/openacs-mail.html
```

- Email addresses
- References to people in the community, such as

```
donb
lars
```

- Code-like terms. These are terms that looked similar to a label in the source code. Examples include

```
ns_cache
apm_package_id_from_key
bboard
```

Such terms will, henceforth, be referred to as *code terms*.

- Noun Phrases. Using natural language processing mixup rules, noun phrases were extracted. Examples include

```
the system
the course
acceptance tests
a number of errors
traceable
postgres
tcl support
```

To illustrate the generation of metadata for natural language text, consider the following message taken from an OpenACS bug report:

```
The URL for a file-storage folder contains I18Nized strings and
tcl pages don't localize them. Furthermore, some of the links
were
missing @?version_id=@contents.live_revision@. Fixed that.
```

The textual portion of this message would be parsed in this stage to extract the following metadata:

- *code terms*:
  - `file-storage`
  - `version_id`
  - `contents.live_revision`
- *noun phrases*:
  - folder

  - I18Nized strings
  - tcl pages
  - localize

3.3.3 Issues in Automatic Metadata Generation

Automatic generation of metadata, especially induced solely by the data itself is necessarily shallow in its scope and prone to errors. By necessity, we are generating metadata after much of the Web data has already been created. This leads to certain complications. For example, we equate the identity of an object with its form. Thus, two different objects of the name `tree_id` will be considered to be the same object. This is correct for the local scope of one object, but globally there may be several objects with the same name. This is particularly problematic for local variables.

In order to correctly identify the scope for all the software objects, we require a parser that understands the software organization and software languages used in OpenACS/dotLRN and can interpret the scope of OpenACS software objects correctly. As discussed in Section 3.3, this involves significant work. Since the objective of Dhruv is to simply present meaningful information rather than an accurate model of the software, we do not distinguish between different objects and leave it to the human reading this information to interpret it correctly.

Due to the imprecise nature of the automatic metadata generation, the resulting extracted information contains a fair amount of noise. This includes

- *extraction errors*: Extraction errors are primarily cases where the extraction procedure identified information that does not occur in the Dhruv knowledge base and therefore cannot be linked to it. There are two possible reasons why such errors occur. It may be that the extraction procedure incorrectly extracts information that cannot then be linked to the knowledge base. It is also possible that the object was not identified and stored in the knowledge base in the first place, once again due to the limitations of the extraction procedure, as discussed above. Finally, a not infrequent reason is that people do make mistakes in spelling an object. While humans are relatively tolerant of such mistakes, our automated system currently cannot be as tolerant.

- *ambiguous context*: When code terms or file references are given without adequate explicit context, it is often difficult to determine which object is being referred to. For example, in the case of file references, there are multiple files called `index.adp`. If people were to read the message where the reference to `index.adp` occurs, they might be able to use information in the rest of the message to determine which `index.adp` is being referred to. This is quite difficult to do automatically. Similarly, the variable `community_id` occurs within the code base as

  ```
  communities.community_id
  non_member_classes.community_id
  non_member_clubs.community_id
  parent.community_id
  subgroups.community_id
  ```

  Determining which of these is being referred to is non-trivial. For the time being, we assume that `community_id` could refer to any of these and report all the possible instances of `community_id` found (see the found-as heuristic in Section 3.4.2).

- *stopwords*: Common words are typically ignored in the field of information retrieval and we follow this practice. This includes the standard stopwords: articles, such as ‘the’, conjunctions such as ‘and’ and ‘therefore’ etc. Our stopword list was drawn from several stopword lists on the Web. In addition, we included a number of stopwords specific to our domain, such as ‘login’, ‘bug’ and ‘pst’. These were added manually to improve the noun phrase extraction process. We were relatively conservative in adding words to the stopword list to avoid losing valuable terms. Similarly, to avoid conflating words and decreasing precision, we did not perform stemming. The full stopword list used in included in the Appendix.

This noise is removed or cleaned up, as applicable, before the extracted information in used in the next stages.

3.4 Linking Metadata

Having generated metadata for the community information space, using the procedures described in section 3.3, we now attempt to *cross-link*[2] the extracted metadata. A cross-link between two objects in the community information space indicates that the objects are semantically related. Each object forms a part of the context required to understand the semantics and role of the other object. This is a fairly general notion and there are obviously different kinds of cross-linkages. For example, between code terms and artifacts, between artifacts and the people who participated in them and so on. A cross-link is similar to a Web link, except that Web links are between web pages and cross-links are between community artifacts and objects.

There are essentially two kinds of metadata extracted in the previous section:

1. Terms, namely code terms or noun phrases, and
2. Artifact references, namely references to bug and patch reports, discussion threads, files and people.

Dhruv determines cross-links between related artifacts using several heuristics, which we discuss in detail below. These heuristics are used to construct a cross-linkages page for each term (code term or noun phrase), which provides details of the term and its links with other artifacts. Cross-linkages for each kind of metadata are determined differently. In the following sections, we describe how cross-linkages are determined for noun phrases (Section 3.4.1), for code terms (Section 3.4.2) and finally for artifact references (Section 3.4.3).

[2]The term 'cross-link' is usually used to denote a side bond that links two adjacent chains of atoms in a complex molecule[Cro]. In this document, we use it to denote links between semantically related objects in the complex community information space.

Figure 3.9: Overview of the process of determining similarity

3.4.1 Identifying Artifacts from Noun Phrases

The *text-similarity* heuristic compares the text of two documents and if they are similar to a degree, it creates cross-links between them. The text of various artifacts were determined differently and drawn from the parsing step described in Section 3.3. For bug and patch reports, and forum discussions, the text of each artifact was taken by concatenating the text of the messages in the discussions. For source code files, the text of the artifact was termed to be the concatenation of comments in the files, both for functions and namespaces, as well as inline comments. For CVS logs, the text of the artifact was termed to be the concatenation of the text of the commit logs for the file.

A common paradigm used in information retrieval is the vector space model, which represents documents as vectors of terms. The set of all documents under consideration, is referred to as the *corpus*: $C = \{D_1, \ldots, D_n\}$. In a corpus that uses m terms, $t_1, \ldots, t_m$, a document is represented by a vector in an m-dimensional space as follows:

$$D_i = (w_{i1}, w_{i2}, \ldots, w_{im}) \tag{3.1}$$

where w_{ij} is a weight representing the importance of term t_j in representing the concepts of document D_i.

The metadata for each OpenACS artifact is extracted as code terms and noun phrases. The artifact is then transformed into a vector with weights for the metadata terms. There are several ways of calculating weights for the terms of a document vector. Dhruv uses the *tf/idf* method [], where the weight w_{ij} is calculated as a product of a local weight l_{ij} of the term t_j in the document D_i, and the global weight g_j of the term t_j in the corpus. The weight w_{ij} is then simply l_{ij}/g_j.

The local weight factor l_{ij} weights terms which appear more often in a document more highly, as they contribute more to the meaning of the document. The local weight is calculated as:

$$l_{ij} = \frac{1}{2} + \frac{1}{2}(\frac{tf_{ij}}{mtf_j})$$

where tf_{ij} is the term frequency, or number of occurrences, of term t_j in document D_i and mtf_j is the maximum term frequency of term t_j in all the documents of the corpus.

The global weight factor g_j weights terms that occur only in a few documents more highly than terms that occur frequently throughout the corpus. This is because the terms that occur less often are more useful for discriminating documents from the rest of the corpus. The global weight g_j is calculated as:

$$g_j = log(\frac{|D|}{df_j})$$

where $|D|$ represents the size of the corpus (number of documents) and df_j represents the number of documents that contain the term t_j.

Given the vector representation of documents, we can now determine their similarity by comparing their document vectors. Dhruv uses the vector-space cosine measure common in information retrieval:

$$sim(D_i, D_j) = \frac{D_i \cdot D_j}{\parallel D_i \parallel \; \parallel D_j \parallel}$$

Vectors D_i and D_j are document vectors derived from Equation 3.1.

This relatively straightforward model does have certain drawbacks. The primary disadvantage of this method is that the document vectors are closely tied to the specific terms used in the documents. The vector space model will not find two documents to be similar, even if they are about the same concepts, if they use different terms to refer to them. In addition, multiple words may have the same meaning (synonymy) and a single word may have multiple meanings (polysemy), which further reduces the accuracy of this method. There are alternate methods, such Latent Semantic Analysis (LSA) that improve on this model by taking into account words that tend to occur together in the corpus. However, since our goal is primarily to explore using various kinds of paradigms in service of the

Semantic Web, in this proof-of-concept work, we use the standard model rather than any particular extension of it.

When searching for artifacts related to a term, we treat the term as a very small document and search for related artifacts by relevance based on their degree of similarity to the search query.

3.4.2 Identifying artifacts from code terms

Given a code term identified through information extraction, we build cross-linkages in two ways:

1. by identifying the type of software object it refers to and
2. by treating it as a noun phrase and using the procedure discussed above in Section 3.4.1.

Different kinds of cross-linkages are created for depending on the type of software object in question. Table 3.1 gives an overview of Dhruv's usage of various heuristics to determine cross-linkages for different types of software objects. The rows represent the heuristics and the columns represent various types of software objects. For reasons of space, the `code:` prefix has been omitted from the software object classes. In addition, the software object `code:OACSPackage` has been abbreviated to OACS Pkg in the table. Similarly, `code:OraclePackage` has been abbreviated to OPkg.

In the following, we go through each heuristic and explain it in detail:

Found in

The *found-in* heuristic realizes the intuitive reasoning that the file a software object has been found in is an important part of the context for that software object. This kind of location heuristic is fairly general and can be applied to most software objects. It is not used for terms of the classes `code:OACSPackage` and `code:Function`. For OpenACS

Heuristic	OACS Pkg	File	Variable	Function	OPkg	Table	Namespace	Query
Found in		⋆	⋆		⋆	⋆	⋆	⋆
Found as	⋆	⋆	⋆	⋆				⋆
Contains functions					⋆		⋆	
Contains variables				⋆				
Contains namespaces	⋆							
Refers to		⋆		⋆				
Has author		⋆		⋆				
Defined in file				⋆				
Defined in package				⋆				
Functions in same file				⋆				
Has bugs and patches		⋆						
Recent committers		⋆						
Requires packages	⋆							
Required by packages	⋆							
Package maintainers	⋆							
Package description	⋆							

Table 3.1: Cross-linkage heuristics for each type of software object

packages, location is not a meaningful characteristic. The only container object above packages is the OpenACS system itself. For `code:Function`, the found-in cross-links are realized through more specific heuristics, defined-in-file and refers-to, which will be discussed presently.

For code terms that occur fairly frequently across the code base, this heuristic yields a huge list of files. It can therefore be less helpful in these cases.

Found as

As discussed in Section 3.3.3, there may be several terms with the same name and extracted code terms may have ambiguous context. For this reason, it is useful to cross-link a term with other terms of the same name. The *found-as* heuristic does precisely this.

In addition, software objects that are semantically related are often given similar names. The *found-as* heuristic harnesses this to identify similarly-named software objects throughout the code-base. For example, the variable `tree_id` is found-as the following software objects:

```
get_mapped_subtree_id
old_tree_id
subtree_id
target_tree_id
tree_id
```

All of these refer to identifiers of a tree in the software code. Such information can be useful when trying to understand the ways in which the variable is used or to find related variables.

Contains Functions

An OpenACS namespace of type `code:Namespace` is a container for numerous functions of type `code:Function`. Knowing which functions are contained in the namespace help in understanding the scope of the namespace. The *contains-functions* heuristic connects namespaces to the functions they contain.

Contains Variables

The *contains-variables* heuristic links a function to the variables it contains. Variables provide more insight into what a function does short of actually reading the definition of the function. For example, a function with variables

```
community_id
```

probably provides functionality that varies depending on the community sin question.

Contains Namespaces

An OpenACS package may contain several namespaces that implement the functionality of the package. Knowledge of these namespaces aids understanding of the package and allows people to go from the package level to focus on individual namespaces. The cross-linkages page for namespaces includes a list of the functions they expose using the *contains-function* heuristic. Together with the *contains-namespaces*, this allows a person to drill down from packages to individual functions that implement the functionality of the package.

Refers to

In OpenACS, `code:Files` and `code:Functions` are both types of code blocks and contain code with control flow. The *refers-to* heuristic attempts to capture the control flow of these code blocks by linking a function or file with the code block it calls. This allows objects linked by control flow to also be cross-linked in Dhruv. Thus, information about which objects may be affected by changes to a particular object are also captured in the cross-links.

Defined in file

Knowing the file and the package a function is defined in is a key part of understanding the definition and use of the function. The *defined-in-file* and the *defined-in-package* heuristics create cross-links, connecting functions to the files and packages that contain their definition. Since Dhruv maintains all files with package name, the defined-in-package heuristic is superfluous if the cross-links from the defined-in-file heuristic is also used.

Functions in same file

To illustrate the usefulness of this cross-link heuristic, let us consider the function `forum::message::close` in the file `messages-procs.tcl` in the `forums` package. In addition to this function, the file contains the following functions:

```
forum::message::new
forum::message::open
forum::message::edit
forum::message::delete
forum::message::get
forum::message::set_format
forum::message::set_state
forum::message::reject
forum::message::approve
forum::message::get_attachments
forum::message::do_notifications
forum::message::subject_sort_filter
forum::message::initial_message
```

Knowing that these other functions exist provides better context to interpret the functionality of `forum::message::close`. For example, we now know that this function does not delete a message, since there exists a specific `forum::message::delete`, which presumably implements that functionality. Thus, the list of functions in the same file provides a comprehensive overview of the kinds of actions that are possible on a message and what portion of those actions is implemented by the function in question, `forum::message::close`. It also gives an indication of how the forum message functionality is organized in files, because other functions in the same `forum::message` namespace may be stored in a different file. The individual functions are likely to be semantically related to each other since they perform related, but complementary actions.

Has bugs and patches

The *has-bugs-and-patches* heuristic identifies the bug reports and patch reports associated with the file. Such information can provide a history of the previous problems with the code and consequent changes made to the file. This provides some background as to why the code in the file is implemented as it as and what kinds of problems can arise as a result of bugs in the code of the file. Similar information is explicitly given by people when they refer to bug and patch reports in the commit comments of a file. This heuristic uses such information in the CVS commit logs, but, in addition, attempts to divine such information automatically from patch and bug reports and references in the files themselves.

Has author

The *has-author* heuristic creates a cross-link between a file and its authors. A person is deemed to have authored a file, if his or her name appears in the file comment, as in the following fragment:

```
@author Ben Adida <ben@openforce.biz>
```

Knowing the author of a file and their expertise can be very useful to understand the nature and quality of the code in the file. It also allows people to know whom to contact in case of question about the code.

Recent committers

The *recent-committers* heuristic is similar in its aim to the has-author heuristic, but focusses instead on the last five people to commit to the file. This can often be more useful than the has-author heuristic, particularly if the file is relatively old and many people have modified it since its creation. The recent-committers heuristic is implemented by examining the CVS commit logs.

Package maintainers

The *package-maintainers* heuristic is similar to the previous two heuristics, but works on the more general level of an OpenACS package. A package maintainer is responsible for receiving bug reports for the package and is, in general, responsible for the package. Often people who have authored the majority of the files in a package take the responsibility of package maintainer (also known as the package owner). Not all packages have a maintainer. Information about package maintainers is usually available publicly. In the case of the OpenACS community, the package maintainer information is taken from the list on

`http://openacs.org/projects/openacs/packages/`

Requires and required by packages

The two heuristics *requires-packages* and *required-by-packages* both attempt to capture the cross-links that arise from package dependencies. The use of this heuristic is similar to the refers-to heuristic, in that changes to one package may affect that packages it is required by. Conversely, a given package may be affected by changes in packages it requires. Package dependency information is determined from the `package-name.info` files in each package's directory.

Package description

The *package-description* heuristic is not really a heuristic in the sense of the other ones presented. It associates a short description of the package functionality and intent with a package. When presenting information about a package, this package description, gathered automatically from package source code files, can be very useful to place the other information in context.

3.4.3 Identifying artifacts from artifact references

Inter-artifact links

During the course of normal activities and interactions within the community, community members tend to refer to other artifacts that explain a point or provide background information for it. Inter-artifact linkage takes many forms:

- CVS commit logs refer to the bug reports that the commit fixes. They also often refer to a patch report whose patch is being committed. Less often, they may refer to forums discussions that contain the discussion that prompted the commit.

- Bug reports in the OpenACS Bugtracker have a slot for the patch reports that address the corresponding bugs. In addition, bug reports may refer to other bug reports that are either duplicates or report similar problems. Bug reports also refer to forum discussions, for example, when the bug was discussed on the forums prior to filing an official bug report. Similarly, bug reports also refer to code files, for example, when describing the problem experienced and the suspected offending files.

- Similarly, forum discussions may refer to files, bug and patch reports, other forum discussions and CVS commit logs, as the need arises.

- Patch reports and files have significantly less inter-artifact linkage, but they may also link to files, bug and patch reports, forum discussions and CVS commit logs.

The artifact references are primarily of the form given in Section 3.3.2 and the procedure given there is primarily used to identify them. The only additional case is for the links between patch reports and files. The files modified in a patch are often not visible in the patch report itself. However, each patch report usually has an associated patch fragment, which consists of the files modified and the changes made. To identify the files referred to by the patch report, the corresponding patch fragment was separately parsed.

Co-authorship

In order to identify people with related expertise, we examine their co-authorship of artifacts. If people tend to participate in the same bug bug reports and discussions and modify the same code files, albeit across different periods of time, then it is quite likely that they will have similar expertise. Thus co-authorship of artifacts has implicit in it shared expertise.

If we consider every two people who co-author an artifact to have interacted, then the co-authorship heuristic essentially determines the professional 'social networks' of the community.

People with similar co-authorship of documents and artifacts are identified using cosine similarity as described for determining text similarity in Section 3.4.1. The only difference is that instead of comparing document vectors with m terms, here we use people vectors with m artifacts.

$$P_i = (w_{i1}, w_{i2}, \dots, w_{im}) \tag{3.2}$$

where $w_{ij} \in \{0, 1\}$ and w_{ij} represents whether person P_i authored artifact a_j.

3.5 Generating Recommendations

Dhruv utilizes the cross-links determined in Section 3.4 to generate recommendations for each message in a bug report. The recommendations for each message are generated on the

Artifact	Authorship
Bug report	Open Report
	Edit Report
	Comment on Report
	Resolve Report
	Close Report
Patch report	Open Report
	Edit Report
	Comment on Report
	Accept Patch
	Reject Patch
	Close Report
Discussion thread	Open Thread
	Comment on Thread
Source code file	Author File
	Modify File
CVS log comment	Modify and commit File

Table 3.2: Nature of authorship on artifacts

basis of the content of the message, i.e. its extracted metadata, namely code terms, noun phrases and artifact references. The artifacts referred to by the artifact references and by cross-links from the extracted metadata are accumulated and then pruned to generate the recommendations. Each type of cross-link is given a recommendation weight to reflect its importance in determining the recommendations. Artifacts are weighted according to the cross-link that picked them. Two different cross-links may suggest the same artifact. In this case, the artifact weights are summed up to give a new recommendation weight to the artifact.

In addition to the cross-links determined in Section 3.4, Dhruv uses a couple of heuristics specifically for generating bug report message recommendations. These are discussed individually in the next section. Finally, the recommendations are ranked according to descending weight and the top n recommendations are finally presented to the user.

3.5.1 Heuristics for generating recommendations

Bug Report Summary

The bug report summary is usually the most informative text in the entire bug report. In the OpenACS Bugtracker, bug reports are summarized by various bug attributes and the bug summary. The summary may determine who and how many people view the bug report and consequently how soon it gets fixed. A bug report with an ambiguous summary is less likely to elicit a response from the community than a well-written, focussed bug report. It is therefore in the interest of the person submitting the bug report to make the bug report summary as informative and topical as possible. The bug report summary is also relatively concise, at most a sentence clause, so each word within the summary is all the more informative. There are times when the bug report summary falls short of being clear and informative, but it is still the most reliable information available to an automated process.

Therefore, artifacts that are cross-linked to by metadata captured from the bug report summary are given a higher recommendation weight.

Within-Message Artifact References

Bug report messages sometimes contain artifact references to other bug reports, files or discussion threads. In such cases, the person posting the message has already determined a related artifact. Personally endorsed artifacts are likely to be more reliable than the artifact references generated by Dhruv. Therefore, such within-message artifact references are given a high recommendation weight.

Cumulative Recommendations

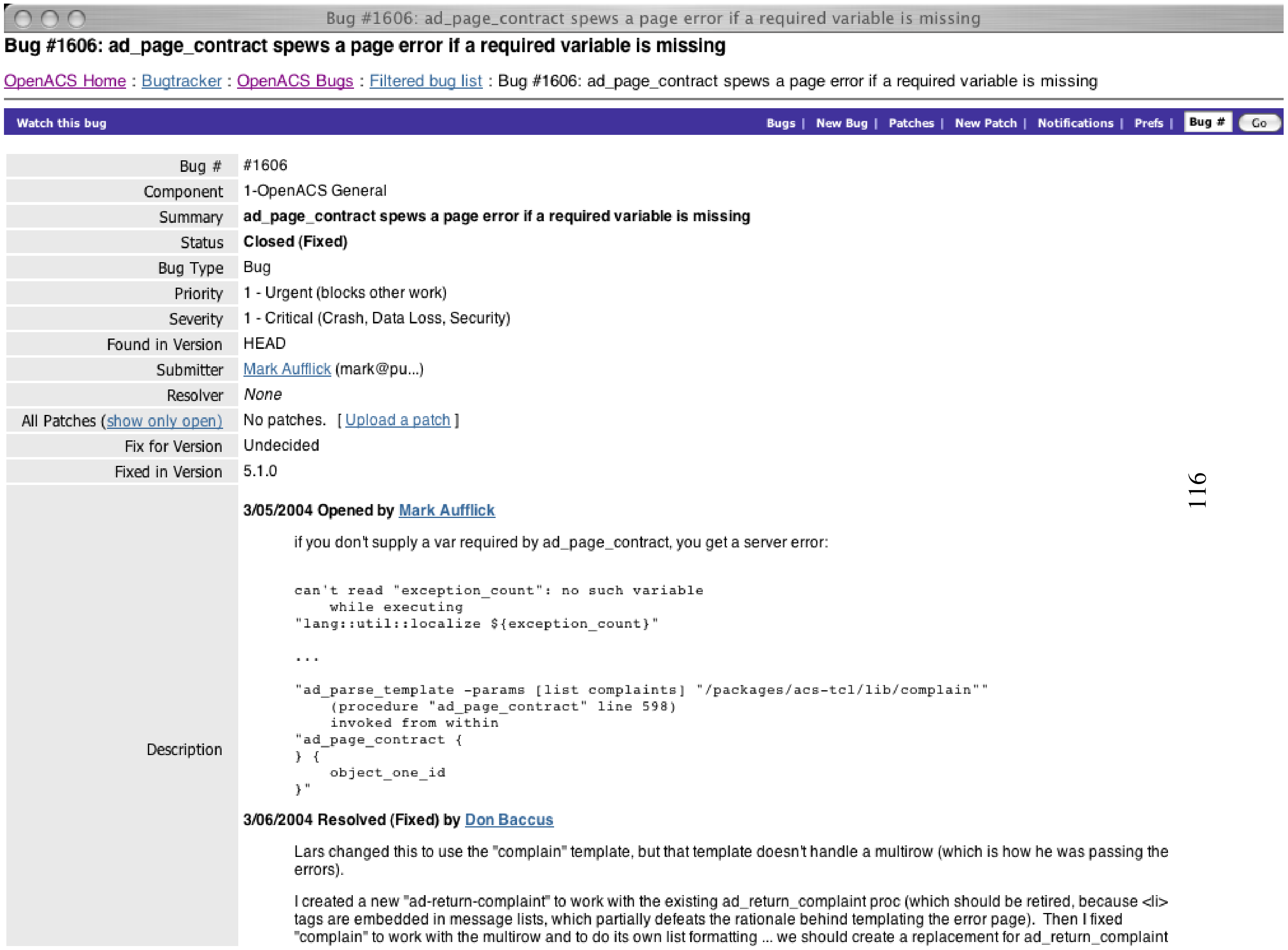
Each bug report message adds a certain modicum of information to the bug report. The semantics of the bug report can only be determined by examining the entire bug report. Thus, in order to give recommendations for the most recent bug report message, Dhruv utilizes the metadata of all previous messages in the bug report.

3.6 User Interaction with Dhruv

In this section, we describe how the various information sources and heuristics are combined in Dhruv to present an enhanced bug report interface for the community. In particular, we discuss the requirements for the enhanced bug report interface and how it determines the Dhruv interface in Section 3.6.1. A tour of the various kinds of HTML pages provided by Dhruv are also presented in the section. We finally present briefly the procedure for generating the interface in Section 3.6.2.

3.6.1 How to use Dhruv?

In this section, we explore the use of the Semantic Web in the context of OSS bug resolution. In particular, we are interested in how OpenACS community members can make use of the cross-links developed by Dhruv for bug resolution. We therefore attempt to integrate Dhruv with the existing OpenACS Bugtracker (Figure 3.10) as smoothly as possible.



Bug #1606: ad_page_contract spews a page error if a required variable is missing

Bug #1606: ad_page_contract spews a page error if a required variable is missing

OpenACS Home : Bugtracker : OpenACS Bugs : Filtered bug list : Bug #1606: ad_page_contract spews a page error if a required variable is missing

Watch this bug — Bugs | New Bug | Patches | New Patch | Notifications | Prefs | Bug # Go

Bug #	#1606
Component	1-OpenACS General
Summary	**ad_page_contract spews a page error if a required variable is missing**
Status	**Closed (Fixed)**
Bug Type	Bug
Priority	1 - Urgent (blocks other work)
Severity	1 - Critical (Crash, Data Loss, Security)
Found in Version	HEAD
Submitter	Mark Aufflick (mark@pu...)
Resolver	*None*
All Patches (show only open)	No patches. [Upload a patch]
Fix for Version	Undecided
Fixed in Version	5.1.0

Description

3/05/2004 Opened by Mark Aufflick

if you don't supply a var required by ad_page_contract, you get a server error:

```
can't read "exception_count": no such variable
    while executing
"lang::util::localize ${exception_count}"

...

"ad_parse_template -params [list complaints] "/packages/acs-tcl/lib/complain""
    (procedure "ad_page_contract" line 598)
    invoked from within
"ad_page_contract {
} {
    object_one_id
}"
```

3/06/2004 Resolved (Fixed) by Don Baccus

Lars changed this to use the "complain" template, but that template doesn't handle a multirow (which is how he was passing the errors).

I created a new "ad-return-complaint" to work with the existing ad_return_complaint proc (which should be retired, because <li> tags are embedded in message lists, which partially defeats the rationale behind templating the error page). Then I fixed "complain" to work with the multirow and to do its own list formatting ... we should create a replacement for ad_return_complaint

Figure 3.10: Bug report #1606 in the OpenACS Bugtracker

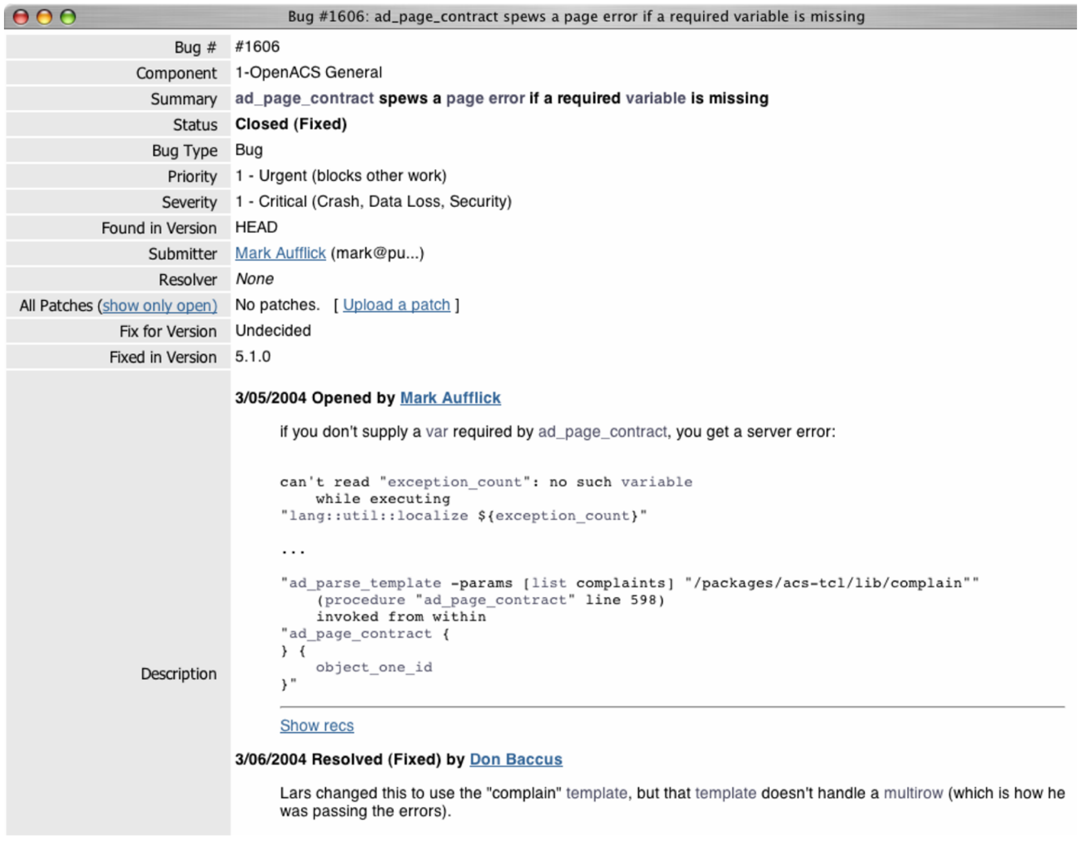
If Dhruv is to fit into the normal process of community bug resolution, then it should support answering the kinds of questions that normally arise in in the minds of developers in the midst of bug resolution. Two kinds of questions that often arise when developers attempt to understand a program are 'what' and 'why' questions [Let86]. The 'what' question represents questions of the form 'What is this software object' or 'What does this software object do?'. The 'why' question represents questions about the purpose and rationale for sections of source code, such as 'Why is this fragment of code implemented in this particular way?'. Dhruv can support both these kinds of questions in the context of bug report comments. The former by presenting the definition of the object and its cross-links. The latter by providing cross-links to the discussions about the software object.

Both 'what' and 'why' questions are posed as soon as the unfamiliar software object is encountered [Let86]. The unfamiliar software objects in a bug report message are captured as code terms, as described in Chapter 3. To support the immediate answering of 'what' and 'why' questions, we turn the extracted code terms and noun phrases themselves into HTML links in Dhruv that developers can click on, as shown in Figure 3.11.

The links highlighted gray-blue in Figure 3.11 lead to more information on the highlighted text, namely the information in the cross-links page for the selected term. The lightly colored links lets people be aware that this term is a special kind of link, so that they can delve further if they wish to. However, the color is muted, so that people reading the bug report may ignore them and are not distracted by the links.

The basic concept of the interface is similar to that used in the KIM front-end [KPO+03]. KIM is a semantic annotation platform, that automatically annotates pages with respect to a given knowledge base and allows people to browse the pages. Annotated text is transformed into a hyper link which leads to a information page about the text. The information on the page is gathered from the knowledge base. The difference between Dhruv and KIM is essentially in the kinds of links highlighted and the kind of information presented.

KIM focusses on annotating domain-independent semi-structured text, such as people, places, dates etc. In Dhruv, the focus is on annotating domain-specific structured code terms and noun phrases. The process of clicking on links to understand more about a bug report is also more in keeping with a focused OSS community and its specific community


Bug #1606: ad_page_contract spews a page error if a required variable is missing

Bug #	#1606
Component	1-OpenACS General
Summary	**ad_page_contract spews a page error if a required variable is missing**
Status	**Closed (Fixed)**
Bug Type	Bug
Priority	1 - Urgent (blocks other work)
Severity	1 - Critical (Crash, Data Loss, Security)
Found in Version	HEAD
Submitter	Mark Aufflick (mark@pu...)
Resolver	*None*
All Patches (show only open)	No patches. [Upload a patch]
Fix for Version	Undecided
Fixed in Version	5.1.0
Description	

3/05/2004 Opened by Mark Aufflick

if you don't supply a var required by ad_page_contract, you get a server error:

```
can't read "exception_count": no such variable
    while executing
"lang::util::localize ${exception_count}"

...

"ad_parse_template -params [list complaints] "/packages/acs-tcl/lib/complain""
    (procedure "ad_page_contract" line 598)
    invoked from within
"ad_page_contract {
} {
    object_one_id
}"
```

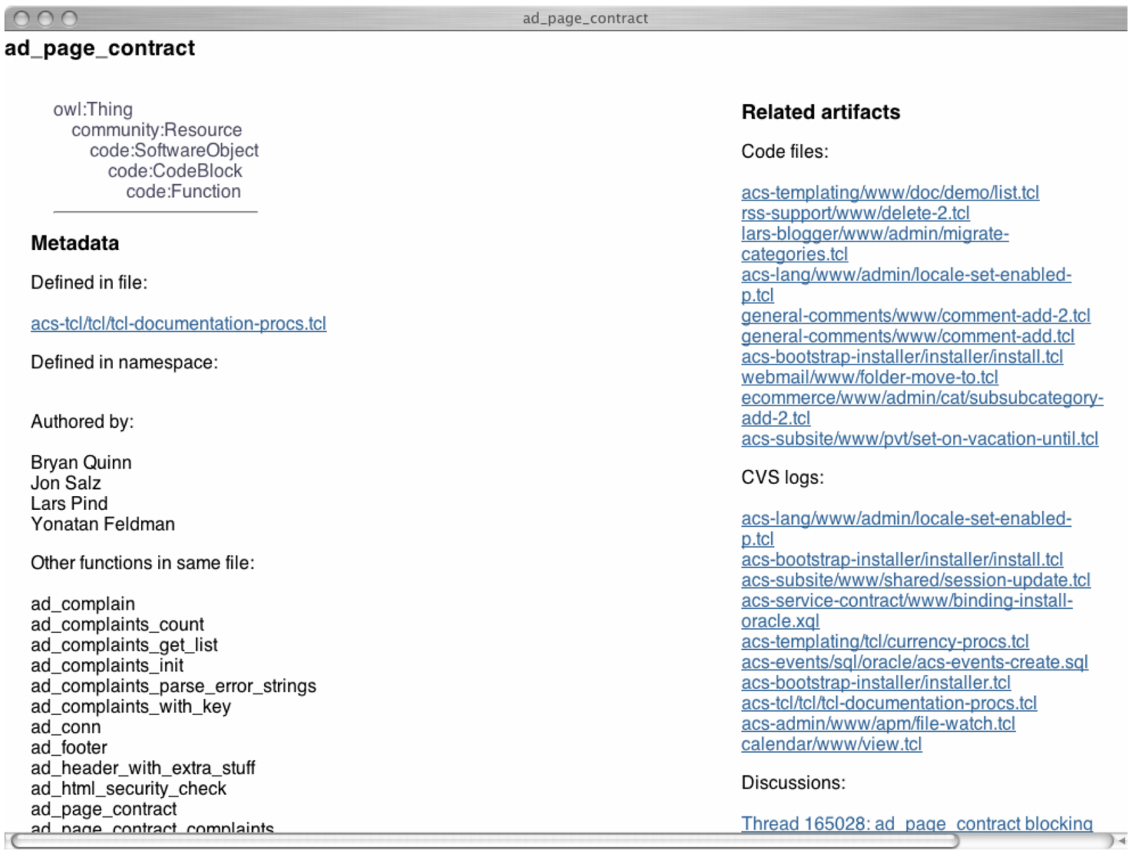
Show recs

3/06/2004 Resolved (Fixed) by Don Baccus

Lars changed this to use the "complain" template, but that template doesn't handle a multirow (which is how he was passing the errors).

Figure 3.11: Dhruv's version of OpenACS Bug #1606

processes.


ad_page_contract

ad_page_contract

owl:Thing
community:Resource
code:SoftwareObject
code:CodeBlock
code:Function

Metadata

Defined in file:

acs-tcl/tcl/tcl-documentation-procs.tcl

Defined in namespace:

Authored by:

Bryan Quinn
Jon Salz
Lars Pind
Yonatan Feldman

Other functions in same file:

ad_complain
ad_complaints_count
ad_complaints_get_list
ad_complaints_init
ad_complaints_parse_error_strings
ad_complaints_with_key
ad_conn
ad_footer
ad_header_with_extra_stuff
ad_html_security_check
ad_page_contract
ad_page_contract_complaints

Related artifacts

Code files:

acs-templating/www/doc/demo/list.tcl
rss-support/www/delete-2.tcl
lars-blogger/www/admin/migrate-categories.tcl
acs-lang/www/admin/locale-set-enabled-p.tcl
general-comments/www/comment-add-2.tcl
general-comments/www/comment-add.tcl
acs-bootstrap-installer/installer/install.tcl
webmail/www/folder-move-to.tcl
ecommerce/www/admin/cat/subsubcategory-add-2.tcl
acs-subsite/www/pvt/set-on-vacation-until.tcl

CVS logs:

acs-lang/www/admin/locale-set-enabled-p.tcl
acs-bootstrap-installer/installer/install.tcl
acs-subsite/www/shared/session-update.tcl
acs-service-contract/www/binding-install-oracle.xql
acs-templating/tcl/currency-procs.tcl
acs-events/sql/oracle/acs-events-create.sql
acs-bootstrap-installer/installer.tcl
acs-tcl/tcl/tcl-documentation-procs.tcl
acs-admin/www/apm/file-watch.tcl
calendar/www/view.tcl

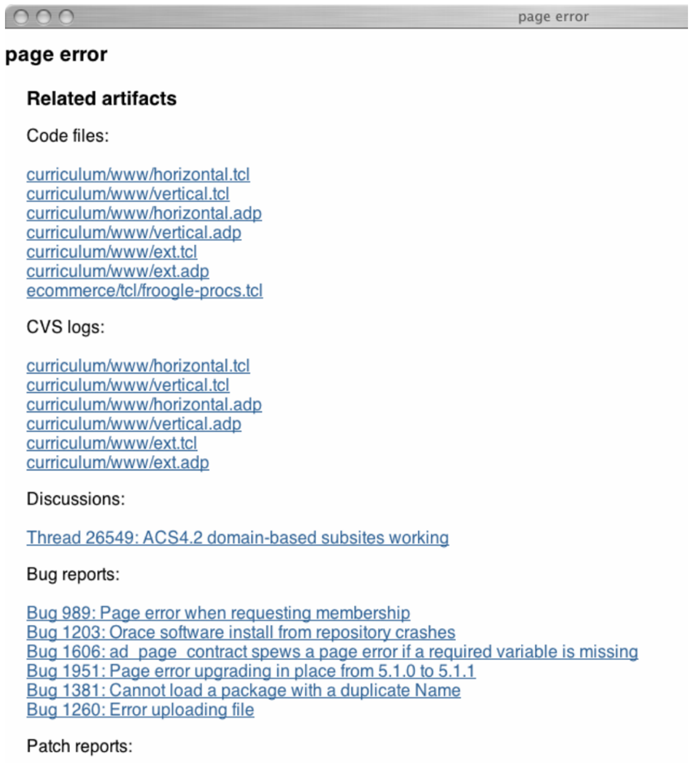
Discussions:

Thread 165028: ad_page_contract blocking

Figure 3.12: Cross-links page for a code term

The cross-links page, as shown in Figure 3.12, lists various kinds of information about the code term, both the semantic in the knowledge base and the cross-links of the term in the community semantic web. Since there are two kinds of terms: code terms and noun phrases, we have two separate kinds of pages for each. Figure 3.12 displays the cross- links page for a code term, where Figure 3.13 shows the cross-links page for a noun phrase. Other kinds of extracted metadata, such as file names, which have corresponding semantic information in the knowledge base, are treated in the same way as code terms.

Structured metadata drawn from the knowledge base is displayed in the left column,



page error

page error

Related artifacts

Code files:

curriculum/www/horizontal.tcl
curriculum/www/vertical.tcl
curriculum/www/horizontal.adp
curriculum/www/vertical.adp
curriculum/www/ext.tcl
curriculum/www/ext.adp
ecommerce/tcl/froogle-procs.tcl

CVS logs:

curriculum/www/horizontal.tcl
curriculum/www/vertical.tcl
curriculum/www/horizontal.adp
curriculum/www/vertical.adp
curriculum/www/ext.tcl
curriculum/www/ext.adp

Discussions:

Thread 26549: ACS4.2 domain-based subsites working

Bug reports:

Bug 989: Page error when requesting membership
Bug 1203: Orace software install from repository crashes
Bug 1606: ad_page_contract spews a page error if a required variable is missing
Bug 1951: Page error upgrading in place from 5.1.0 to 5.1.1
Bug 1381: Cannot load a package with a duplicate Name
Bug 1260: Error uploading file

Patch reports:

Figure 3.13: Cross-links for a noun phrase

while the right column displays related artifacts of various kinds. The related artifacts are categorized into different kinds of artifacts, such as code files and bug reports, because different artifacts carry different kinds of information about a given code term that a bug fixer might need.

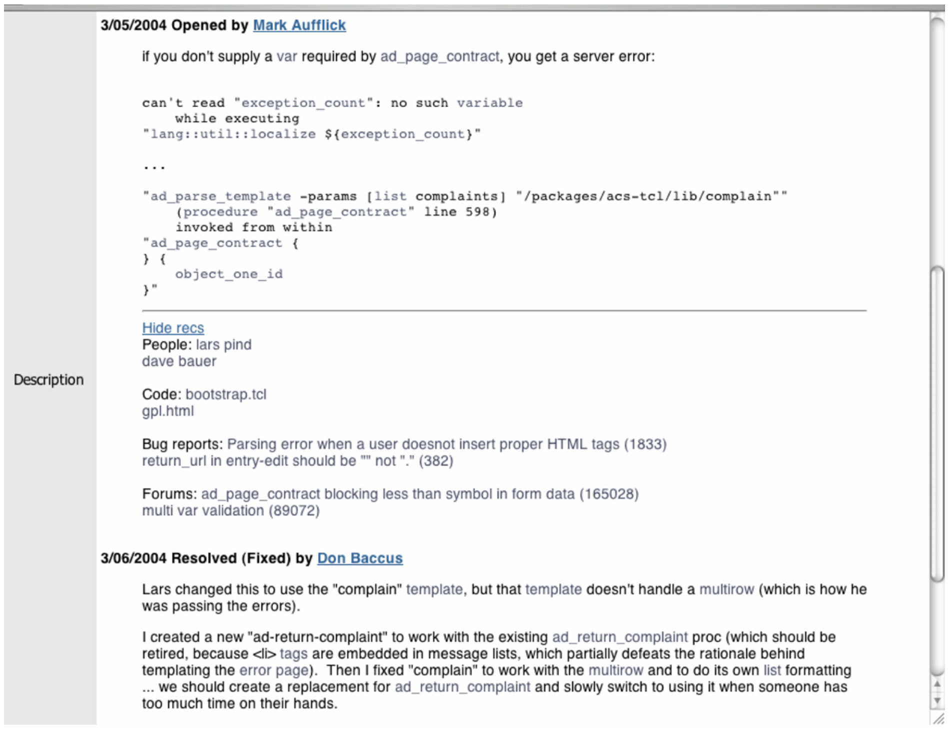
Together, the highlighted links and cross-links pages comprise an enhanced interface to Dhruv's semantic data for individual terms. In addition, Dhruv also presents artifact recommendations for each message. When a community member comments on a bug report, Dhruv produces recommendations of related artifacts for the comment and appends them to the message, as depicted by Figure 3.14. Message recommendations represent key artifacts for the message as a whole, based on the cross-links of individual terms. They can represent a best guess for new developers, who are unsure where to start with the bug report. If a artifact is clearly related to the entire message, then the message recommendations ought to capture it and present a short-cut for developers looking at the bug report.

Figure 3.14 shows the message recommendations given by Dhruv. These recommendations are classified into several categories, just like the related artifacts for a term. The rationale is the same: to let people easily get to the different kinds of information represented by different artifacts.

3.6.2 Generating the Interface

The interface presented was generated by creating cross-links pages for terms and modifying bug report pages, as follows:

1. Construct the Dhruv semantic knowledge base
2. Process bug report messages to extract terms. Create a modified bug report page, where the extracted terms are transformed into links to cross-links pages for the term.
3. If there is no existing cross-links page, create a cross-links page for the term



Description

3/05/2004 Opened by Mark Aufflick

if you don't supply a var required by ad_page_contract, you get a server error:

```
can't read "exception_count": no such variable
    while executing
"lang::util::localize ${exception_count}"

...

"ad_parse_template -params [list complaints] "/packages/acs-tcl/lib/complain""
    (procedure "ad_page_contract" line 598)
    invoked from within
"ad_page_contract {
} {
    object_one_id
}"
```

Hide recs
People: lars pind
dave bauer

Code: bootstrap.tcl
gpl.html

Bug reports: Parsing error when a user doesnot insert proper HTML tags (1833)
return_url in entry-edit should be "" not "." (382)

Forums: ad_page_contract blocking less than symbol in form data (165028)
multi var validation (89072)

3/06/2004 Resolved (Fixed) by Don Baccus

Lars changed this to use the "complain" template, but that template doesn't handle a multirow (which is how he was passing the errors).

I created a new "ad-return-complaint" to work with the existing ad_return_complaint proc (which should be retired, because <li> tags are embedded in message lists, which partially defeats the rationale behind templating the error page). Then I fixed "complain" to work with the multirow and to do its own list formatting ... we should create a replacement for ad_return_complaint and slowly switch to using it when someone has too much time on their hands.

Figure 3.14: Message Recommendations

4. Based on the cross-links for each extracted term in a message, generate message recommendations and incorporate them into the modified bug report.

Chapter 4

What is the potential impact of the Semantic Web?

In this chapter, we present the evaluation of the Dhruv prototype, whose creation was presented in Chapter 3 and whose use was presented in Chapter 3.6. We first discuss how the potential impact of the Semantic Web can be measured in Section 4.1. We use two different ways to evaluate Dhruv: a comparison of Dhruv's recommendations with historical data and a qualitative user study to evaluate Dhruv's information interface. We discuss the advantages and limitations of these evaluation methods in Section 4.2. In Section 4.3, we present the comparison of Dhruv's recommendations with historical data. In particular, we explore the effect of various cross-links heuristics on Dhruv's recommendations in Section 4.3.3, the effect of learning links on Dhruv's recommendations in Section 4.3.4, the contribution of the text similarity heuristic in Section 4.3.5 and finally Dhruv's recommendations for explicitly related bug reports in Section 4.3.6. We then move on to the users' assessment of Dhruv's information interface in Section 4.4. In particular, we describe how the study was conducted in Sections 4.4.1–4.4.2. In Section 4.4.3, we present the user feedback for individual components of Dhruv.

4.1 How is the potential impact of the Semantic Web measured?

Having constructed a prototype Semantic Web for open source software communities in the form of Dhruv, we can now determine the potential impact of the Semantic Web on online communities and interactions. The impact of the Semantic Web is measured by evaluating the extent of support Dhruv can provide in resolving an individual bug. In keeping with the exploratory nature of this research, we follow a multi-pronged approach to evaluating the potential impact of Dhruv. In constructing Dhruv, we have focused on two aspects of the Semantic Web, providing an enriched interface and providing recommendations for each bug report message. We evaluate each of these in turn:

1. Dhruv's recommendations are evaluated qualitatively for a sample of bug reports and their usefulness assessed.

2. Using the historical logs of past instances of bug resolution in the OpenACS community, i.e. the past bug reports, as a guide, we determine whether Dhruv can recommend some of the artifacts that were actually referred to in the bug report. Historical data is the best source of information about bug resolution processes within the community.

   In particular, we examine how individual heuristics and data sources affect the quality of Dhruv's recommendations. We use 'localized probes' to examine Dhruv's suggestions.

3. An important component of the evaluation is to have OpenACS community members' assessment of Dhruv and the enriched information interface it presents on the OpenACS community. This is done via a qualitative study, involving a think-aloud where community members examine an actual OpenACS bug report using Dhruv and answer questions in a structured interview.

   A qualitative study with a think-aloud component allows us to capture a wide range of feedback, opinions and experiences of OpenACS community members using

Dhruv. This is important to evaluate all aspects of Dhruv and identify the major directions for future work. A think-aloud test involves having the study participant use the system while 'continuously thinking out loud' [Lew82]. The constant verbalization of the participant's thoughts allows the researchers to get a direct understanding of how the participant views the system, what parts of the interaction are problematic and the major confusions of the participants [Nie93]. This is particularly important given that we wish to evaluate the information interface and information flow presented by Dhruv rather than more concrete user interface aspects of the system.

We are also constrained by time and the availability of people. OpenACS being an online community, its members are dispersed globally, just like most other OSS communities. Thus, a face-to-face study can only involve a small number of OpenACS community members. Furthermore, bug resolution is an involving and time-consuming activity. It cannot therefore be conducted in a set amount of time. The think-aloud study allows us to gather a lot of qualitative data from a small number of users in a relatively short session.

4.2 Discussion of the measures

We do not have any gold standard for evaluating Dhruv's recommendations besides the historical data of the OpenACS project. Some of the resources involved in the resolution of the bug are linked in the bug report. These include other bug reports (especially duplicate bug reports about the same bug), relevant discussions, files and the people involved in the resolution of the bug. Unfortunately, the historical data available is an incomplete representation of the entire resolution of the bug. The most completely recorded resources are people. Besides people, the historical data for bug reports is meager. Only about a third (1/3) of the bug reports have links to any artifacts. Among these, the best represented artifacts are source code files. Bug report messages tend to mention filenames when reporting technical information on bugs. In addition, patch reports contain information about which file was modified and thus was relevant to the patch report. Thus, if a bug report is associated with a patch report, then we can associate files with the bug report. The least number of artifact links are to discussions. To compensate for the scarcity of available historical information, we will use modified measures to evaluate Dhruv's recommendations, as discussed in the next section.

We can be sure that the artifacts linked to in the bug report are somehow involved in the resolution of the bug report. However, some of the artifacts links may be incidental and the artifacts themselves only tenuously relevant to the bug report. For instance, a bug report may link to another bug report, which is not directly related, but ought to be resolved at the same time. However, manual examination of the corpus reveals that such cases are relatively rare.

Even if the historical data were richer in its artifact links, even then comparing Dhruv's recommendations to the links in bug reports would only present part of Dhruv's usefulness in the bug resolution process. This is because bug reports do not capture the entire bug resolution process as it occurs. Rather what they capture can be better described as intermittent snapshots of the process. Sometimes, intermediate steps in the process are not represented in the bug report at all. For example, a bug report comment may describe the bug in high-level and ambiguous terms. A developer might then analyse the bug, track down the problem, devise a solution and submit a patch, while only commenting in the bug report that the bug has been fixed. If these were two consecutive bug report messages, it would be infeasible for Dhruv to give any relevant recommendations. In comparison, chat logs better capture the bug resolution activity as it occurs. Developers conduct detailed discussions of bugs and code implementation in the chat rooms. Unfortunately, these discussions can be unfocussed, so analysing the support Dhruv can provide to bug resolution interactions in the chat rooms is non-trivial. Dhruv would need to know when to start suggesting artifacts and when to keep silent. It would require, for example, a separate analysis and segmentation of chat room logs into bug resolution sessions.

In comparing Dhruv's recommendations to historical data, we try to analyse the *correctness* of Dhruv's recommendations. However, the community semantic web envisioned in Chapter 2 focusses on providing *meaningful* information. Correct information will be meaningful to the community resolving the bug report, but not all meaningful information may be correct. In the sense that it would be directly used in the resolution of the bug. Thus, a direct comparison of Dhruv's recommendations to the historical data would underestimate the number of meaningful or useful recommendations. In addition, there is a danger that people (the gold standard) might not use the optimal artifacts for the resolution of bug. Thus, the links they use may not be 'correct'. This would reflect poorly on Dhruv's performance, since Dhruv tries to find the best artifact from its point of view. To minimize this danger, we ask people to rate the quality of Dhruv's recommendations.

To evaluate the usefulness of Dhruv's recommendations, we present them to members of the OpenACS community and ask them to judge the value of the recommendations. This is done during the qualitative study of OpenACS community members using Dhruv. The qualitative study of OpenACS community members is appropriate for several reasons. First, Dhruv is a proof-of-concept prototype rather than a full implementation. Thus, since Dhruv as a concept is still developing, a formative evaluation is more appropriate than a summative one. Thus, instead of measuring the performance of people using Dhruv, the qualitative study will allow us to focus on evaluating the information flow of Dhruv's interface rather the detailed usability of the interface as a whole. It will bring Dhruv's ease of use and the difficulties and confusing aspects of its information flow to the fore.

4.3 Evaluating Dhruv's recommendations

Using the historical data logs as a guide, we examine whether Dhruv can recommend some of the artifacts that were actually referred to (and therefore required) within a bug report. Essentially, we would like to know whether Dhruv can find the same kinds of things that human processing by the community would reveal later on. In the following, we examine the effects of various artifact cross-links heuristics and recommendation heuristics. Of course, this is a large space. We present localized probes to a way to examine the influence of certain heuristics on the behavior of Dhruv on historical information.

We validate Dhruv recommendations by evaluating whether the recommended resources for a bug report were actually required later on in fixing the bug. For bug reports with links to resources (namely discussions, people, bug reports, patch reports, code files), we:

1. determine recommendations for each message,
2. identify resources actually mentioned by people in the bug report, and
3. for each resource found, determine if it is contained in the recommendations of earlier messages.
4. Determine the precision/recall of the recommendations.

We did not take the bug report links to patch reports into account. In a random sample of 2/3 of such links, the patch reports linked to were the patch reports that actually fixed the bug report.

4.3.1 Corpus

We drew our evaluation corpus from the existing set of OpenACS bug reports. We picked five OpenACS packages with the most resolved bug reports, namely the following packages: `acs-lang`, `acs-subsite`, `acs-tcl`, `acs-templating`, `dotlrn` and `fs`. For each package, we constructed a training set and a test set from the bug reports for the package. Bug reports were randomly assigned to one of the sets, such that 70% of the bug reports were used in the training set and 30% in the test set. The training and test sets for each package were then accumulated into one training set and one test set respectively. Altogether, there were 235 bug reports in the training set and 100 bug reports in the test set.

Of the 100 bug reports in the test set, there were 276 references to people. In other words, there were on average about three different people participating in the bug reports of the test set. Similarly, there were 113 source code file references, 4 references to other bug reports and 1 reference to a discussion. The references to source code files include files identified through patch reports associated with the bug report.

In the full set of OpenACS bug reports that we drew from, the total number of bug reports were 2370, with 824 links to files, 85 links to bug reports, 158 links to discussion threads.

4.3.2 Statistics measured

We evaluate the quality of Dhruv's suggestions using precision and recall statistics for each of the three categories: files, bug reports and discussions. Precision measures the proportion of good recommendations to all recommendations, whereas recall measures the proportion of all recommendations that were good. We will give more precise definitions to this intuitive interpretation below.

Our set of known relevant items is minuscule compared to the number of recommendations. Every message has at most one or two relevant items of each category, but it gets at least ten recommendations in each category. Thus, measuring precision as a ratio of relevant items to items recommended will not be an appropriate measure. It would grossly underestimate the quality of results in terms of bringing up something relevant. Instead, we measure precision in terms of whether every set of recommendations contains the relevant items. Thus, the precision P is calculated as

$$P = \frac{n_c}{n_r}$$

if n_c represents the number of times that Dhruv gave a correct recommendation and n_r represents the number of times that bug messages had known relevant items that could be recommended. if B_r is the set of known relevant items for a bug report message and d_r is a recommendation from Dhruv, then

$$n_c = \sum_{i=0}^{m} p_i$$

where

$$p_i = \begin{cases} 1 & \text{if } d_r \in B_r, \\ 0 & \text{otherwise.} \end{cases}$$

In addition, most people will scan the results and further process and filter them. The precision statistic above measures in how many of the cases where some artifact could be recommended, is the artifact actually recommended. If the set of recommendations is very large with respect to the known relevant items, then this number will approach the recall measure. It does not measure how much of the recommendations are useful results. That will be assessed by potential users of the system during the qualitative study.

Recall is computed in the standard way for each category of artifacts. Recall measures the proportion of artifact links actually recommended by Dhruv to the artifact links it could have recommended. So, if n_i represents the number of artifact links correctly identified by Dhruv and n_m represents the number of artifacts that the system missed, then the recall R is computed as

$$R = \frac{n_i}{n_i + n_m}$$

As mentioned previously, the test set has relatively few links to bug reports and discussion threads. Due to this, the values for precision and recall tend to fluctuate in the experiments below. Even for file recommendations, the files actually used during a bug resolution may be affected by information not contained in the bug report itself. Thus, it is difficult for any tool to completely predict the files that will be used in the resolution of a bug report. the values below are closer to the lower bounds for the precision and recall of such a tool. Since the mapping of a term to a file is also not always clear, the values for these measures are always on the lower side.

4.3.3 Comparing Cross-links Heuristics

The recommendations given by Dhruv depend significantly on the weights assigned to different components of Dhruv. These components essentially are the various heuristics for creating cross-links. The effect of varying weights for the different components ranks some heuristics higher than others, which in turn affects the quality of Dhruv's recommendations.

Here, we measure the precision/recall of Dhruv's recommendations using various heuristics. In each case, only the cross-link type being tested has non-zero weights; weights for the other types of cross-links are set to zero. For each cross-link heuristic, we generate 10 and 40 recommendations and measure the quality of the full set of recommendations. The sets of 10 and 40 recommendations represent small and medium sets of recommendations. By evaluating two sets of recommendations for each type of heuristic, we can see how the heuristic ranks the good recommendations. Ideally, a good heuristic or good combination of heuristics would give high precision/recall scores for the small set of recommendations and the increase in precision/recall from 10 to 40 recommendations would be relatively low. This would indicate that the good recommendations are contained in the top 10 recommendations.

The precision and recall of Dhruv's recommendations using various single heuristics is shown in Table 4.1. We compare Dhruvs recommendations for five heuristics: text similarity (see Section 3.4.1), index links, inter-artifact links (see Section 3.4.3), learnt links and found-in links (see Section 3.4.2). The learnt links are essentially inter-artifact links isolated within the training set of bug reports. Inter-artifact links are created through the ongoing interactions of the community. Thus, new inter-artifact links are constantly being created. Although inter-artifact links are gathered over the entire history of the project, the learnt links allow us to examine the effect of a small number of additional inter-artifact links on Dhruv.

Clearly, using no weights, i.e. not ranking the results at all, does not produce very good recommendations. The 10 and 40 recommendations are an arbitrary subset of cross-linked artifacts, resulting in low precision and recall.

Among the heuristics, the inter-artifact links and found-in links are best in identifying relevant artifacts, especially for a small set of recommendations. It is somewhat surprising that inter-artifact links perform so well, since they essentially enhance existing recommendations. After getting a first set of related files from other cross-links heuristics, the inter-artifact links build on these recommendations to find other files referred to by the

	10	40
No weights	0.09/0.04	0.09/0.04
Learnt links	0.15/0.07	0.22/0.12
Text Similarity	0.15/0.07	0.39/0.19
Index links	0.20/0.10	0.57/0.28
Inter-artifact links	0.33/0.16	0.57/0.43
Found In links	0.39/0.19	0.50/0.24

Table 4.1: Precision and Recall for Code Files Using Various Heuristics

recommendations. The inter-artifact heuristic uses only one hop of related files, but this one hop significantly improves results.

Although found-in links heuristic leads to the best recommendations for the small set of recommendations, it performs noticeably less well for the large set of recommendations. This is probably because the found-in links heuristic relies solely on code terms being present in the message. If there are some code terms in the message, then the heuristic is able to give highly accurate recommendations. Without any code terms, though, this heuristic is useless and therefore, cannot perform well on its own. The entire limited set of its contributions is anyway concentrated in the top 10 recommendations, yielding high precision. Thus, additional recommendations do not yield significantly better results.

Another interesting result shown in the table above is that text similarity on its own is relatively poor at eliciting good recommendations from Dhruv. This suggests that for some reason, the textual content of an artifact does not reflect the semantic content of the artifact. It may be that there are too many documents using similar words, but referring to different conceptual ideas. Or that the semantics inherent in the semi-structured nature of the artifacts, such as CVS commit logs and bug reports are not captured within the text of the artifact. Thus, other sources of information, such as those captured by Dhruv are required.

Cross-links created by text similarity and by learning from previous bug reports do result in good recommendations, at least in isolation. Index links give somewhat better information, but not as much as inter-artifact links. This may be because there is insufficient code term information in the bug reports. Given that inter-artifact links give good information, learnt links may not add much because they are not comprehensive. They only cover the bug reports in the training set.

We next examine the influence of combinations of these heuristics on Dhruv's recommendations. In each case, all heuristics in a combination are weighted equally. The results are shown in Table 4.2. To understand which links are more informative, found-in or inter-artifact links, we examine them in various combinations (lines (a)–(e)).

	Text Similarity	Found In	Learnt	Index	Inter-artifact	10	40
(a)			⋆		⋆	0.37/0.19	0.52/0.27
(b)		⋆			⋆	0.44/0.21	0.59/0.37
(c)		⋆		⋆		0.43/0.20	0.67/0.32
(d)		⋆	⋆			0.48/0.23	0.52/0.27
(e)		⋆	⋆		⋆	0.48/0.23	0.63/0.41
(f)	⋆	⋆				0.35/0.17	0.67/0.32
(g)	⋆				⋆	0.31/0.15	0.65/0.33
(h)		⋆	⋆	⋆		0.44/0.21	0.74/0.35
(i)	⋆	⋆	⋆	⋆		0.56/0.27	0.76/0.38
(j)		⋆	⋆	⋆	⋆	0.54/0.27	0.78/0.44
(k)	⋆	⋆	⋆	⋆	⋆	0.67/0.32	0.80/0.40

Table 4.2: Precision and Recall for Bugs, Discussions and Code Files Using Various Links

Comparing combinations (a) and (b) with the results of inter-artifact links alone in Table 4.1, we see that both found-in and learnt links improve on the recommendations generated by inter-artifact links, found-in more than learnt links. On the other hand, comparing combinations (b) and (c) with the results of using the found-in heuristic alone (Table 4.1), we see that both inter-artifact links and index links improve on the recommendations generated by the found-in heuristic, the inter-artifact links more than the index links. This is unsurprising, since the found-in and inter-artifact heuristics both seem to be individually informative. The above observations indicate that the two heuristics carry different kinds of information.

Learnt links improve on the recommendations generated by the combination of found-in and inter-artifact links (line (b)). However, adding the inter-artifact heuristic does not improve the recommendations generated by the found-in and learnt links heuristics together (lines (d) and (e)). This is fairly surprising, since it suggests that found-in links and learnt links provide complementary information. Furthermore, the recommendations generated by them together contains the relevant recommendations generated by inter-artifact links.

Adding recommendations by text similarity links reduces the quality of recommendations generated by either the found-in heuristic or the inter-artifact links heuristic, as indicated by lines (f) and (g). However, this effect primarily occurs for the smaller set of recommendations, indicating that text similarity does not affect the actual recommendations as much as it changes their relevance ranking inappropriately. This may be because similarity of their texts does not determine the relevance of artifacts for fixing bug reports. The text similarity heuristic may also perform less well because we have used only the textual comments and in-file documentation as a basis for judging similarity for source code files. Once again, combining text similarity with the found-in heuristic does better than combining text similarity with the inter-artifact links heuristic.

The last four combinations (lines (h)–(k)) examine the effect of adding text similarity or inter-artifact links to the heuristics trio of found-in, index links and learnt links. The three heuristics together perform as well as the combination of found-in and inter-artifact heuristics (lines (h) and (b)). Adding both text similarity and inter-artifacts heuristics improve the recommendations generated by the three heuristics (lines (h), (i) and (j)). The text similarity heuristic provides more complementary information and therefore, increases the precision/recall of Dhruv's recommendations more. However, the effect of the text similarity heuristic is less for larger sets of recommendations, again suggesting that text similarity primarily influences the ranking of the top recommendations.

Finally, using all five heuristics produces significantly better recommendations than any subsets of the heuristics (line (k)). This indicates that each heuristic provides slightly different information and need to all be considered together to provide the best recommendations. Determining the actual heuristic weights that result in the best recommendations is left for future work.

The most striking result is that none of the cross-link heuristics are able to predict the discussions that bug report messages refer to in the top 10 recommendations. Given that there is only one reference to a discussion thread in the test set, this might a case where the test set is not able to give us fine-grained information about the quality of recommendations. With 40 recommendations, Dhruv is able to predict the discussion In contrast, for bug reports, Dhruv less consistently identifies the correct bug report. thread every time.

4.3.4 Adding learning or evolution of semantic web

The OpenACS community's usage of Dhruv can implicitly provide Dhruv with new connections between artifacts. As people interact and use Dhruv, they provide Dhruv with training data to learn new cross-links from. Dhruv can track people's usage by logging clicks or by gathering explicit links from bug reports to other artifacts. These 'learnt links' are essentially the same kind of data gathered by inter-artifact links for bug reports. Here, we are interested in examining how such additional links affect Dhruv's recommendations. This allows us to investigate how the Semantic Web might change and respond as a result of learning from people's activities and usage of the Semantic Web.

The effect of increasing the weight of the learnt links heuristic when generating recommendations on the precision and recall of the recommendations is shown in Table 4.3. Learnt links are essentially links in the messages of the bugs reports in the training set that are used when generating recommendations for bug reports in the test set. In addition, the links between bug reports and files gathered via the patch reports are also included in the learnt links.

Keeping the weights of other heuristics constant[1], one would expect that increasing the weights for learnt links would improve the precision and recall of the recommendations. Learnt links help identify which links are stronger or better, because they have already

[1]at 5 for the text similarity and inter-artifact heuristics and at 1 for the found-in and index links heuristics.

appeared in the training set of previous bug report resolution instances. This is borne out by Table 4.3.

#recommendations	10	40
0	0.44/0.21	0.72/0.43
5	0.48/0.25	0.76/0.38
7	0.48/0.25	0.74/0.37
10	0.52/0.27	0.69/0.35
15	0.56/0.28	0.67/0.34

Table 4.3: Precision and Recall of File Recommendations For Various Weights of Learnt links

The small set of recommendations steadily improves in quality as the learnt links heuristic is weighted higher. However, the larger set of recommendations decreases in quality as the recommendations supported by the learnt links are ranked higher at the expense of more relevant recommendations. This indicates that the learnt links heuristic can be potentially useful for improving precision of recommendations. It is therefore more useful for fewer recommendations than for a larger set of recommendations. The relatively small jump in the quality of recommendations from the small to the large set for higher weightage of learning links indicates that learning links tend to pull more relevant links higher up.

As discussed previously, learnt links contribute less to predicting bug reports and discussion threads. Within 10 recommendations, learnt links is unable to predict any bug reports. For a larger set of 40 recommendations, the learnt links heuristic is able to identify the discussion thread and one of the four bug reports. Since there are relatively few references to bug reports and discussion threads, it is not surprisingly that relatively little is learnt to improve their recommendations.

4.3.5 Text Similarity

In this section, we examine the effect of increasing the weight of the text similarity heuristic on the quality of Dhruv's recommendations. We previously examined how the text similarity heuristic performed as a singleton and in combination with other heuristics, where it seemed to provide novel information not provided by the other heuristics. Does increasing the weight of the text similarity heuristic in combination with other heuristics improve the recommendations? The results are presented in Table 4.4.

Text Similarity weights	File precision/recall
5	0.48/0.23
10	0.31/0.15
15	0.31/0.15
20	0.28/0.13
25	0.14/0.07

Table 4.4: Precision and Recall Vary with No. of Recommendations for Code Files Using Text Similarity

Increasing the weight of the text similarity heuristic does not improve Dhruv's recommendations for files. On its own, there seems to be a limit to the usefulness of text similarity. This suggests that the other cross-links heuristics capture information and semantic connections that are not reflected in the text of the documents. This may be because we treat the text of a source code file to be the concatenation of its comments. Some files may have inadequate comments and thus are not well suited for the text similarity heuristic.

For source code files, we therefore need the extra semantic connections captured by the metadata and additional cross-links. The other cross-links, in particular inter-artifact links, represent semantic connections made explicitly by people and therefore much more reliable. Surprisingly enough, there seem to be enough of these links to be exploited successfully and improve Dhruv's recommendations.

4.3.6 Explicitly related bug reports

Bug reports that are explicitly marked as being related by OpenACS community members provide another dimension to explore Dhruv's operation. The primary kind of explicitly related bug reports are bugs marked as duplicate bugs. Dhruv ought to suggest each bug report as a related artifact for the other. In addition, Dhruv should suggest related artifacts for the two bug reports.

We selected all the bug reports closed as being duplicates. From these, we created a mapping between bug reports and their duplicates. Duplicate bug reports that did not refer to the 'original' bug report were discarded. Using this mapping, a training and test set were created. The duplicate bug reports were assigned to the test set and the 'original' bug reports to the training set. We then used the links in the training set to generate recommendations for the bug reports in the test set and measured the precision/recall of the recommendations. We further examined how closely related the recommendations for the first message of each pair of the related bug reports are.

	Normal test and training set		Exchanged test and training set	
	Learnt	Text Similarity & Learnt	Learnt	Text Similarity & Learnt
Bugs precision/recall	0.6/0.6	0.7/0.7	1/0.8	1/0.8
Disc. precision/recall	1/0.5	1/0.5	0.33/0.25	0.33/0.25
File precision/recall	1/1	1/1	0.86/0.75	0.71/0.62

Table 4.5: Precision and Recall For Explicitly Related Bug Reports

The precision and recall for related bug reports is shown in Figure 4.5. The first two columns display the results of using the links in the training set to generate recommendations for the test set. The last two columns display the results of performing the same experiments, but with exchanged test and training set. The rationale for this is that usually the bugs marked as duplicate are substantially less rich in message content than their 'original' bugs. Thus, duplicate bug reports may be submitted after the original bug report or they may provide less information about the bug than the so-called 'original' bug report.

Given related bug reports, Dhruv is able to use the learnt links heuristic to predict which files and discussions will be required in the duplicate bug report. Given the asymmetry in the richness of the 'original' and duplicate bug reports, Dhruv is able to use the learnt links heuristic to identify relevant files. It performs less well with discussions, but surprisingly well with bug reports.

The text similarity heuristic decreases both precision and recall of Dhruvs recommendations. This may seem somewhat surprising given that in Section 4.3.3, the text similarity heuristic improved the performance of Dhruv in combination with other heuristics. However, in that section, we examined the effects of combinations of heuristics for a random set of bug reports. Here, we consider a non-randomly sampled set of bug reports, bug reports that explicitly related. It is not surprising then that the learnt links heuristic performs particularly well for these bug reports. The text similarity heuristic reduces Dhruvs precision/recall probably because of the asymmetry in the content of a pair of related bug reports. In addition, duplicate bug reports filed for the same bug often use different language to describe the bug, making it difficult for the text similarity heuristic to perform well.

Qualitative examination of the recommendations for related bug reports also reveals that the asymmetry in duplicate bug reports can be of several kinds. One bug report may discuss the bug and its solution, whereas the other duplicate bug report may present the steps required to cause the bug to manifest itself and the expected and actual behavior of the system (OpenACS bug reports #1465 and #1449, and bug reports #1450 and #1485). The only commonality is that they both refer to a function that does not exist in the code base (`weblogger_channels__name(integer)`). Dhruv's recommendations are based on message content and therefore, unsurprisingly, examination of Dhruv's recommendations for the two bug reports reveals them to be considerably different.

On the other hand, when messages are very similar, Dhruv's recommendations can also be very similar (e.g. OpenACS bug reports #902 and #903). Two bug reports were filed by mistake and both have the same message. In this case, Dhruv suggested six common bug reports and nine common discussion threads.

Similar messages do not always get similar recommendations. For example, OpenACS bug reports #1591 and #1512 are about the same bug and are filed by the same person. In each case, Dhruv identifies the other bug report as relevant, but suggests no other common artifacts.

As the related bug reports accrue more messages, Dhruv's recommendations for the two bug reports can become more similar. The first messages of OpenACS bug reports #1474 and #1378 are considerably different, in that the former has a lengthy explanation of the bug, whereas the second contains a one-line description of the bug. Dhruv thus recommends only one common bug report and no common discussions or files. However, if Dhruv's recommendations for the last message of the two bug reports are compared, 5 of the 10 files recommended are common and 2 of the 10 discussions.

In summary, Dhruv is able to recommend artifacts and resources for bug report messages with varying degrees of precision and recall. The inter-artifact links are the most useful heuristic in isolation. However, the heuristic on its own does not give as good recommendations as the combination with other heuristics, namely text similarity, found-in links, index links and learnt links. The inter-artifact links heuristic is a meta-heuristic in the sense that it is applied on a base set of recommendations already gathered as a result of applying other heuristics. Since inter-artifact links capture human links between artifacts, they are able to improve the precision of the base set of recommendations. Given that inter-artifact links represent links between resources just like web links represent links between web pages, it is likely that using web algorithms for mining link structures on the Web will also yield good rankings for self-contained domains where the links are to artifact references.

Although inter-artifact links were most useful for Dhruv in recommending files, text similarity performed better for recommending bug reports and discussion threads than for files. This is probably because they contain less structured terms for the other heuristics to hook on to. Interestingly, using only text similarity in combination with learnt links reduces the power of Dhruvs recommendations for explicitly related bug reports. This is probably due to the unique textual characteristics of explicitly related bug reports, namely asymmetry in richness of content and differences in the words used to describe the bug.

Text similarity is likely to perform better for normal bug reports. Although the text similarity heuristic is useful in combination with other heuristics, Weighting it too highly reduces the quality of Dhruvs recommendations. This suggests that the words used in the text of artifacts is not rich enough to capture the full range of information contained in community artifacts. This is most obviously true for software code, but also holds for other artifacts with semi-structured data, such as bug reports. Ultimately, it is the combination of all the heuristics together that allows Dhruv to make the best recommendations.

The relations described in Section 3.4.2 contribute to the cross-links pages created by Dhruv and thus indirectly the message recommendations. For future work, it would be useful to try to understand which of those relations contribute most to Dhruvs recommendations.

4.4 Evaluating the Dhruv information interface

In order to evaluate the potential of the Semantic Web from the perspective of the OpenACS/dotLRN community, we performed a qualitative study with a number of experienced OpenACS/dotLRN developers. There were two components to the study: a think-aloud where the participants were asked to examine and walk-through a bug report with and without Dhruv, and a structured interview where the participants were asked to assess the current state and future potential of Dhruv. The objective of the study was to probe the community's current usage of the OpenACS Bugtracker and explore the community's potential usage and assessment of Dhruv.

4.4.1 Study Participants

The target population for the study were OpenACS/dotLRN community members in and around Boston, who had participated substantially in the community. Since OSS community members can be located all over the globe, we chose Boston in order to maximize the number of OpenACS developers who could participate in the study in a face-to-face setting. Based on their publicly available community interactions, six potential study participants were identified. These participants included a variety of people, ranging from bug reporters and people who primarily fixed bugs to people who were heavily invested in OpenACS/dotLRN and had more of a monitoring/guiding role in the community. Of the five people contacted, we had four respondents and three people who actually participated in the study.

All three study participants were substantially involved in the community. One of the participants uses the software for his professional as well as leisure activities. He is a veteran of code development and bug resolution in the community, having used the software significantly for around six years. He is also on the dotLRN Technical Advisory Committee. Another participant had been a peripheral member of the OpenACS community for a long time, before getting more substantially involved in the community recently. Both these participants have commit privileges to the OpenACS code. The third participant is on the Board of Directors of the dotLRN Consortium and therefore a primary stakeholder in the community.

4.4.2 Study Tasks

During the study, the subjects were asked to answer questions about a publicly reported bug in the OpenACS/dotLRN project. In order to answer the questions, they needed to use Dhruv's version of the bug report. The subjects were asked to think-aloud as they used Dhruv and their behavior was observed. The observations were also recorded via a screen-recording software and an audio-recording software.

The format of the qualitative study was as follows. The participants were first given a general introduction to the study. The motivation of the study and its objectives were explained to the participants. They were also given a broad overview of Dhruv's operation. A structured interview was then conducted, where the participants were asked several questions to guage their experience with the OpenACS bug resolution process and to determine the kinds of facilities they make use of, such as search or the OpenACS API documentation. They were therefore asked about the extent of their involvement with the Openacs/dotLRN community, how often they tend to work on bugs and how typically go about the bug fixing process.

Then the participants were asked to do a think-aloud as they examined a bug report assigned to them. The bug report was one that they had fixed previously or related to one that they were involved in. They were asked to read through the bug and do a walk-through of the bug report, explaining what the bug is about and how they would approach the bug were they fixing it. In addition, they were asked about how they used other resources, how they interpreted the messages of other community members, which fragments of the message were most informative and how they figured out the location and cause of the bug. The focus was on trying to understand how they currently fix bugs.

The participants were then shown the same bug report on Dhruv and each of the features of Dhruv were explained as they were encountered. The participants were asked to comment on whether the information presented would have been useful to them during the bug resolution process. They were asked to focus on the nature of information and which information presented was particularly useful. They were also asked to comment on the usefulness and correctness of Dhruv's recommendations for the bug report.

The final stage of the study was a structured interview. The participants were asked to comment on Dhruv's function: the choice of highlighted terms, the cross-links pages for these terms and Dhruv's recommendations. They were then asked to for their opinions about the usefulness of Dhruv: are any things specifically liked/disliked or which changes would make Dhruv particularly useful. Finally, they were asked if a system like Dhruv would be useful to have when fixing bugs and whether it fit into their way of working, into OpenACS/dotLRN and in OSS communities in general.

For each of the study participants, an actual bug report in the OpenACS community was chosen. The criteria for assigning bug reports to study participants were manifold. First, the participant had to be substantially involved in the resolution of the chosen bug report. This criterion was chosen so that we could be certain that the participant has some expertise in the area of the bug report. Second, the bug report must have been resolved at least several months before the study. This is to ensure that the participants actually need to use the Web and Dhruv to examine and talk about the chosen bug report. By choosing bug reports resolved several months before the study, it was relatively unlikely that the participant would remember anything but very high-level information about the bug report. The final criterion for assigning a bug report to a study participant was that Dhruv should perform on average on the bug report. When this was not possible for a single participant, we tried to choose bug reports for multiple participants, such that Dhruv performed on average over the multiple bug reports.

Overall, we tried to reach a balance, such that bug reports were neither too difficult for them, for instance because the participants knew nothing about the area of the bug, nor too easy for them, because the bug is so fresh in their minds that they do not need to use Dhruv to talk about the bug report.

4.4.3 Study Results

The comments from study participants are mainly on Dhruv, but they include some comments on the Semantic Web. All the participants were professionals and were interested in Dhruv and the notion of a community semantic web from a technical point of view. Two of the participants already knew about the Semantic Web and therefore commented on Dhruv based on their knowledge of the Semantic Web.

In the following, we have included some of the comments of the participants verbatim. Within the comments, the author's explanations are delimited by { and }.

Highlighted terms

The first component of Dhruv that the participants encountered were the terms that are highlighted and turned into links. The participants were asked to comment on whether they thought the terms Dhruv highlighted were informative and whether they were likely to click on them. Their responses indicate that they generally felt that the highlighted terms were useful.

Upon first encountering the highlighted links, Developer A asked:

Can you automate this? Is it so that it would be automated and it would go

> *through and search ...*

Upon being told that the links were automatically generated:

> *{clicking on the 'limited access' link} This is cool. Limited access. That's neat, because it's a very specific word, because that's something I would actually [click] ... Because when I see 'system' {another highlighted word}, I see something so general, that it might not pertain to this bug. But 'limited access', in this case, is a useful semantic addition, because it's likely that you'll directly find code where ['limited access'] would be in. Pertinent code.*

However, there are only a small portion of specific terms to more general terms. Developer A noted:

> *So, this 'non-guest users'. This would be something that would be useful, because it is so specific and would likely come with actual results in the code. These {along with 'limited access'} are two specific things. Otherwise, I don't really see specific things that are.*

All of the participants tried to click on highlighted code terms, indicating that these are naturally informative terms. In the case of Developer A, Dhruv missed highlighting some of the potential code terms and a filename. Developer A pointed them out:

> *[if code were linked to the code base] that would be nice. That would be very nice. In the same sense that this [pointing to the file name]. I should think that would be low-hanging fruit. You know, if this shows up at all [in the message], then it would be nice if it showed up there [in the recs]. That should be possible.*

In the bug report examined by Developer A, he pointed out that Dhruv highlighted terms correctly in one message, but not in the other. This was seen as undesirable.

Developer C also immediately zeroed in on code terms and started talking about what he expected the terms would link to. However, there were also more generic terms that caught his attention. One of the 'irrelevant' terms highlighted in the bug report for Developer C was 'strange behavior', which intrigued him:

> *'Strange behavior'. That's interesting. That 'strange behavior' is linked. {clicks on the link} I wonder if these all have the word 'strange behavior'. {verifying that it is so} That's fun. I like that. I think that's great. I like this a lot.*

Upon reflection, the study participants felt that the terms highlighted were generally useful. As Developer B expressed:

> *Actually, I would say that in terms of what it's highlighting, the unstructured text that you're pulling out of the comments, it did a really good job of highlighting stuff that might be relevant.*

Developer C concurred and pointed out that terms that were irrelevant could always be filtered out by the person reading the bug report:

> *I didn't see any examples of things it didn't pick up that were important. I certainly saw examples of things that probably weren't important. Based on what I've seen, it didn't pick up totally irrelevant stuff, like the word 'the'. Most of the stuff it picked up was at least unique in some way. Of course, I can still filter it. Like 'strange behavior', I might look at because it's a curiosity, but I don't think that would really be relevant ... I didn't see anything that was really plain and mundane, which would have been the worst case.*

Developers A and B also felt that they would not click on links that looked uninformative.

> *The word ['system'] is so general. I wouldn't even look at it. I could make that decision just by looking. On the other hand, just by looking at 'limited access', I know it's specific enough.*

Developer B pointed out that although he would click on code terms, they might only appear after the bug has been fixed

> *I would probably go for ... I would probably click on something that was directly related to the bug title and probably would just ignore something [obviously unrelated]. .. I would click on the code terms, but the problem is that we have a lot more information now because [a bug report participant] has actually fixed the bug.*

Developer A was cautious about the value of the highlighted links.

> *The ones that are specific enough {like 'limited access'} are [informative]. ... It would be better if it was more specific.*

During the study, the participant had clicked on a non-distinctive word ‘system’ and then was surprised to see very specific references to ‘system’. When asked whether the specific references shown for terms that were general was a useful thing:

> *Yeah, that's good, that things were more specific than expected, when clicked on really general terms.*

Did the participants think the links were distracting in any way?

> *It's certainly unobtrusive. Need to decide what is good enough to link.*

said Developer C.

In summary, most of the time, Dhruv picked up the structured terms, such as file names and code terms, that people thought were relevant. When Dhruv did not pick up on some of the structured terms, people missed being able to click on them. They felt that Dhruv should have been able to get them. This indicates that the terms in the structured text are markers for people too. The terms possess high semantic meaning.

For noun phrases, people felt that highlighted terms that were general, such as ‘community’ and ‘system’ were obviously less informative than more specific terms, such as ‘course admin’. They were more likely to click on the latter as the more specific links are more likely to lead to useful information. The terms that sounded general, when clicked on, led to a cross-links page with many specific instances in Dhruv. However, despite this, people said they simply wouldn’t make use of these terms. This indicates that there is potential for Dhruv to learn from the links that people click. There is also much scope for improving Dhruv’s selection of terms to highlight in this area.

A closely related problem is that of managing expectations. Although every participant recognized the possibility of filtering the links, some of the participants were nonchalant about filtering links, while one participant was less satisfied.

None of the participants thought the presentation of the highlighted links was distracting or obtrusive.

Learning links

The participants were told that Dhruv could potentially learn automatically from links used by people. Developer A noted that users could also be asked directly about whether a link is useful:

> *It would be cool, you know, you were talking about how do we get info back from the users? As people use this. Looking at link paths and stuff might be a little more difficult than adding just, when you go here {clicks on a link} is it useful or not? In that way, you can start weighting things.*

Developer C, on the other hand, felt that it would be better for Dhruv to learn on its own. Or at least give people a choice about whether they wanted to teach Dhruv.

> *I don’t really like to have to explicitly teach computer programs things, because in my experience, it’s just a waste of time. Like spam filters, it never*

learns. ... I'm like, the computer is the tool, not me. ... I'd rather have the links tracked and that adjusts the weights rather than me explicitly having to say. It would be taking up space in my busy screen and my busy life. Maybe it's a preference. The user enables it: I want to be explicit or have it just be implicit in what I click.

Cross-links Pages

Following a highlighted term led the participants to the cross-links page for that term. As the participants explored the cross-links pages, they were asked to comment on how useful they found the information on those pages.

After reaching the cross-links page for 'limited access', Developer A remarked:

And then the bug reports that's neat, because you'll be able to check if there's another bug there that's related. How did you do this search? ... It would be cool if we had something like [Dhruv].

The next time, Developer A explored some of the other recommended links:

So, now going to 'limited access'. I probably get the same results as before, right? {Yes. Clicks on 'limited access' in second message}. Actually that would be interesting to see. {clicks on recommended code file and scrolls through file.} This has a little bit to do with that. We added that `read_private_p` *{code object added by the developer}. So, that's useful just to be able to see that. And the logs {clicks on the recommended log file}. Cool. That's neat.*

Later on, Developer A summarized:

{clicking on limited access} So, [the code files] that's useful. The different types of discussions, so that's useful as well. Because you want to see what

> *people have said in the past when you're looking at the bug. And then you can understand the problem, so that's very useful. But it's only useful in the cases where .. that's why I picked this limited access, because it's very specific. So, if I go here to 'community' [clicks on community link], I know that it's not going to be ...it's going to be all over the place. So, in that case, it's not useful. The more specific [the term] is, the more useful it is, assuming that [Dhruv] follows the same pattern in other reports.*

Developer B used Dhruv to verify a hypothesis about a software procedure:

> {*Looking at* `util_unlist` } *I've never used this procedure before. Looks like it might be in* {*clicks on* `util_unlist` *link*}, *yeah, that's what I was thinking ... [* `acs-tcl` *]* `utilities-procs.tcl` *because that's where this kind of stuff gets dumped. That's kind of cool. That it. It's certainly interesting, looking back at a resolved bug how this thing can be helpful as it pulls different pieces of information together.*

Developer B later summarized that:

> *I would say that I think it is useful. I'm trying to separate out a potential user interface, once you work on the usability stuff, I think it would be useful. ... I would also tend to click on the more directly related terms. And anything is going to be included, picked up by the system based on those terms will have a much higher probability of being useful.*

Within the cross-links pages, the links to CVS logs were thought by many of the participants to be useful. As Developer B explains:

> *For me, a lot of times, when fixing a bug, it would be great if I could get from the bug report a list of the recent CVS logs because then you can see what's happened and you do have that. That would be probably one of the more important [things].*

Developer C felt that the links to files that where code terms were defined were also particularly useful:

> *The defined in file is very useful. I think this is great so far. If all you did was develop the model and guide the spider to get everything, it's pretty impressive.*

On seeing that some of the lengthy lists presented by Dhruv on the cross-links pages, in particular using the found-as and found-in heuristics, Developer B said:

> *I'm not sure I think the [large found-in list] is too useful. It might be nice to see different occurrences, but where do you draw the line? [large list] Like this is obviously not too useful. ... I probably wouldn't use too much of this stuff. {clicking on* `subtree_id` *and pointing to the short found-in list} See here, this is useful. I noticed that before that when there's a few instances here, yeah, being able to click on a variable name and get real quickly the names, the substring of a name and get to that particular file [is useful].*

For the artifacts identified by term-similarity, responses were mixed. Developer C stated for the related artifacts for a noun phrase:

> *{pointing to the recommended code files} Yes the file does exactly, because this is one of the places that defines the relationship {denoted by the noun phrase} in the data model.*
>
> *{pointing to the recommended CVS logs} CVS logs not relevant.*
>
> *{pointing to the recommended discussions} [The first discussion in the list is] definitely related to the bug, but maybe not to the term ['course admin']. The next hmm. It's kind of a U-shape. {referring to the relevance ranking of the list} Very relevant, not relevant, very relevant again.*
>
> *{pointing to the recommended bug reports} The bug report links do look relevant.*
>
> *I think it's great. Do productivity metrics.*

As Developer B noted:

> *Again it has a lot to do with accuracy. Certainly if the data that was being pulled in was accurate ... I do feel that accurate information, that would be pretty useful. I don't think it would hurt.*

As Developer B noted:

> *This is the kind of thing that I felt would be helpful. Having direct links [from the messages] to the potential offending files and some type of a [CVS log] history. But he's kind of put this in.*

Developer C later mentioned a caveat to judging the relevance of a link from its title:

> *And [Dhruv] may actually do a better job than is evident to me, because someone picked a bad title and your algorithms figure out that maybe it is relevant, but the way the person titled it, it didn't say so.*

To summarize, the cross-links pages were generally found to be quite useful. In particular, the links to files and CVS logs were appreciated. Regarding the various kinds of cross-links heuristics used, the participants particularly mentioned the found-in and defined-in heuristics as being useful. A detailed exploration of the value and use of individual heuristics would be useful in identifying the information content of the cross-links pages.

The found-in heuristic sometimes produced lengthy lists of related items for some terms. While the information it listed was correct, the shorter lists were better appreciated. This is because they were thought to be more informative. This suggests that the amount of information available for a term might be an indicator of how informative the term is. This can then be taken into account when deciding whether the term ought to be highlighted.

The related artifacts produced by term similarity produced mixed responses. Most of the artifacts were thought be relevant. Where the relevance was not clear, a participant pointed out, it could be due to the uninformative titles of bug reports and threads. For source code files where the relevance was unclear, one participant spent several minutes searching for the term in the text of the suggested documents. When he couldn't find the term, he remained mystified as to why the document was suggested. This suggests that recommended documents, that don't contain the term, be ranked lower than ones that do contain the term. Or alternatively, there be an explanation for why a given document was recommended.

Message Recommendations

Each bug report message has a number of recommendations for the following types of resources: people, source code files, bug reports and discussion threads. The participants were asked to comment on the message recommendations. Their responses indicate that the message recommendations were not as useful as the cross-links pages. As Developer C explained:

> *The nice thing about links at top were that they didn't really disrupt my flow. These [message recommendations] disrupt my flow. Probably before really digging into those, I'd go into a code buffer and kind of like fool around, 'cause like now I'm not getting content about the bugs. Like here {pointing to message text and the highlighted terms} I'm getting content of the bugs and these are just nice little embedded helpers. Here {pointing to message recommendations} I've got to actually figure out the structure of what's going on. You know, read each item and decide if I want to go after it. I probably want to read through all the comments. It's almost the second round of considering the bug where I guess I would go into stuff like that. I don't feel super-strongly about it. I'd like it to be off by default.*

Developer A added that the recommendations were also only really useful at the beginning of the bug resolution:

> *If you're at the anterior posts, these aren't relevant anymore. There's a point*

where, at the beginning when you're actually searching for things ... to resolve [the bug] ... it's useful.

Given that message recommendations for the last messages of the bug report change less than those for the early messages, this suggests that perhaps recommending artifacts for each bug message is not as useful as recommending artifacts for the whole bug report.

The display of the message recommendations was also distracting. As Developer B explained:

... this is a very terse presentation and it just adding more text that's not directly related to the bug comment or post just complicates what you're looking for.

However, the possibility of dynamically showing and hiding recommendations was appreciated:

The recommendations are very intrusive. You don't want that all the time, probably. ... I like the fact that [the recommendations toggle] makes it easy to hide or show.

So, how useful and relevant were the recommendations themselves? Developer B commented on the people recommendations:

That is definitely a useful piece of information. ... Like if you know what package the problem is in .. Oftentimes, the bug can be in a service-level package. With the forums and irc being as critical for development, if you know a person is a primary resource, if you know their name, a lot of times, you can search for that person's forum history and go through his commits or go online and ask him directly.

However, the people recommended were not as useful specifically for the bug itself. Developer B explained further:

Sure, how much work has [the recommended person] done? I'm sure ... he's touched every piece of the toolkit. He's one of those guys, one of those incredibly productive guys, who sits down and fix four-five bugs in a couple of hours and you see all these commit messages ... who looks through all these random packages. Looks through here, finds a typo.

There is some evidence that the links for each term and links for each message might be confusing for people, at least at first. People look at the cross-links page expecting message-specific recommendations and are then confused by the split between the cross-links pages and the message recommendations.

As Developer C noted:

For example, I would have gone to `forum-security-procs` . *If I click on* `forums::new_questions_allow` , *what I'm going to be looking for is this {points to the defined-in list} or let's do a little test. Looks for* `forum-security-procs.tcl` *in cvs logs for* `forum::new_questions_allow` *and doesn't find it. If you could see the same [metadata] view for a file, that'd be something.*

Figuring out why sometimes I have a little block on the page and sometimes it expands to its own page. It would be useful to know.

Overall, there were mixed responses to Dhruv's recommendations. The majority of the participants felt that the recommendations did not fit in with their method of fixing bugs. In contrast with the highlights links to cross-links pages, which were thought to be useful. The only type of recommendation appreciated by one of the participants was the suggestions for related people.

The majority of the participants were also skeptical of the recommendations given by Dhruv. For example, the people recommendations were thought to be accurate, but not useful. Since Dhruv rates productive people more highly, it suggests people who are prolific coders, but not necessarily related specifically to the bug report at hand. This is one area where the classification of people into various roles and types could be used fruitfully.

Most of the participants also felt that the recommendations should be shown more unobtrusively, if at all. The ‘Show/Hide recs’ toggling of the recommendations was therefore appreciated.

How to improve Dhruv?

The participants freely gave numerous suggestions for improving Dhruv, ranging from minor user interface issues to more fundamental suggestions about the function of Dhruv. The suggestions are presented in detail below:

1. **Make machine-generated aspect salient:** To someone unfamiliar with Dhruv, it may not be clear that the highlighting and cross-linking process, as well as the recommendations are machine-generated. Developer A felt that the user should be made aware of this from the start:

    > *Well, I think this is definitely neat. It would need to be clear .. it would be quite clear to someone who uses the system, but that this is machine-generated, so people are not like what’s this?. That you have some kind of indication that it’s possibly related like ’Possibly Related’. So, when I look at all these [recommendations], these pretty much aren’t related.*
    >
    > *Just making sure that it’s clear that it’s machine generated. You probably just have to modify this text, that little blurb there. And when you mouse over these links, you might just want to have some pop-up there that says linked by these recommendations, cause what the heck are these links? Why are these links showing up? So being able to see why the links are showing up. Possibly related links or some way of indicating .. You could probably do that with CSS or something.*

2. **Improve cross-links display:** There were specific suggestions on how the cross-links displayed could be improved. For example, the found-in heuristic, which links

to files a software object is found in, could inform users of exactly where in the often lengthy file the software object can be found. Developer B:

> *You could include a line number in your grep that would give a person an idea of where in the file you would find [the function/variable].*

There are several kinds of information views of a file that can be useful to developers: the actual source code of the file, the CVS change logs for the file and cross-links for the file. Developer C explains that Dhruv should incorporate links to all of these:

> *It would be really useful if I could see the source code of the file and the CVS logs for the file. There are three places a file link could go to: CVS logs, source code and one to kind of metadata, where you have the neighborhood. Potentially any of those could be useful, so I'd like to be able to get to any of that.*

3. **Improve modeling and cross-links for people:** Different institutions using OpenACS contribute code implementing different functionality. They also tend to focus on fixing bugs in the code they contributed. Therefore, Developer A suggested modeling institutions in addition to people. People within institutions are in some responsible all responsible for the code contributed by the institution and thus 'substitutable':

   > *Now, you're doing people, it might be interesting if you added institutions, because a lot of time code that's added has the copyright of the institutions, so that's how you could tell. And then associating people with institutions, that might be something too.*

   Regarding the cross-links pages, Developer C suggested that having a separate type of page with cross-links for people would be useful:

   > *One thing that occasionally is useful that is not here, is that whenever a person comes up, provide a way to get to other stuff that person has*

> *created. Like in the forums package, there's a view of all the forum messages from the user and that should be unified. Like there should only be page with everything the user did and the [OpenACS] software is not quite there yet. But at least, we've got the forums page.*

4. **Direct Interaction with Dhruv:** There is currently no way to interact directly with Dhruv and its knowledge base, to query it directly or to tell it that something is relevant. Developer A wanted that possibility:

   > *I've always wanted to be able to say, you know, add something to the knowledge base, right? So, it would be neat if I don't understand something to mark it as a key term. Like what the heck is 'limited access' user? What does that mean? What does 'guest user' mean? And just to be able to mark those terms as something that need to be described. And that's why it's useful with that forum posting that crops up. That's very useful and I would immediately, if this functionality was there, and I could say 'yes' that's related or 'no' that's not related, I would immediately click that. You have the related pages, but it's not put to use. And that would be a place where you can map those relations.*

   Explicit user interaction would allow Dhruv to learn from users. If a users, in the natural process of analysing a bug report, mark the terms they don't understand, Dhruv can take that as a signal to create cross-links pages for them. Thus, through this dialogue, Dhruv can create cross-links pages for a more parsimonious and relevant set of terms.

5. **Elicit more information from users:** If Dhruv could elicit more information from users, then it could present contextually appropriate information and give better recommendations. Developer B pointed out that if the OpenACS Bugtracker were to be extended slightly, developers could note the exact web page of the OpenACS user interface where they were experiencing problems within the bug report itself. This would allow Dhruv to use that user input to give tailored recommendations:

> *I think if you could tie in somehow we could pick a page and ... I would probably navigate to the page, where the bug was, the offending [page] and then examine the code on that page. So, somehow, showing the exact page, you could pull up a list of tcl procs using namespace and stuff, what the various other packages and stuff are that the page is calling. If you could show that, that would be something. Maybe, if you pick a particular package, if it's a UI bug, you could show a list of pages it might have occurred on, in a drop-down list, and then you could go out using the cvs browser to kind of grep the procedures that are called. And if you could see that in the bug list, that would be pretty helpful.*
>
> *[Otherwise] What I would have to do is install the [package where the problem is located] and follow these steps [to reproduce the bug]. But if it was listed, the page where the problem was occurring, I may be able to go right there. And what might be involved in this is an hour's work just to set this up, or fifteen-twenty minutes just to figure out what page it is.*

6. **Downplay underlying semantic machinery:** Although Dhruv uses semantic information in the background to produce the cross-links and recommendations, the ontologies are not visible to users. The only place where the underlying ontologies are hinted at is in the top-left corner of the cross-links page. This information was seen as extraneous and therefore potentially confusing. Developer C:

> *One of the first things I noticed on the cross-linkages page, which I think was useful, was the semantic model, even though it powers everything and makes it go, for someone who hasn't spent time thinking about it, it just takes up space. It has deep meaning in terms of how it allowed you to create these views but for me it sort of was just confusing. And maybe if I got really adept to the point where I understood your model, then it'd be useful, I'd at least initially.*

7. **Improve presentation of recommendations:** Finally, many of the participants felt that presentation of the recommendations needed improvement. The difference be-

tween the message recommendations and related artifacts on the cross-links pages was not always clear to the participants. As Developer C expressed:

> *I guess an interesting question is what really is the difference between this block and when you click on this, you see all these related artifacts. What the difference between here and the next page, it's not really clear.*

The extra information displayed for the message recommendations was also distracting for the participants from the primary task of fixing the bug. As Developer B noted:

> *Well, from a UI perspective, just from a very simple, there's just a lot of information. Maybe [the recommendations] can be presented or described in a way that's a little more ... cause this is a very terse presentation and it just adding more text that's not directly related to the bug comment or post just complicates what you're looking for. But if there was an offset, like you have in diff and when it was shown, it was in a different color and offset.*

The 'Show/Hide recs' toggle link could be useful in letting people know that they can access Dhruv's recommendations any time they wish to do so. Developer C recommends:

> *.. Have the hide recs, have that [link] closed, but expandable.*

Does Dhruv fit into the OSS bug resolution process?

The participants felt that Dhruv would fit naturally into their bug resolution process. Specifically, as Developer B comments:

> *I would say that it fits into that natural way of trying to grab at things outwards from a few little kernels trying to take this one and where can I go with this particular piece of code or this comment. It would jive with my kind of approach. I can't speak for others. In terms of developers, there's a wide*

range. For developers who know all, this would be kind of superfluous. And some of those developers who are not exactly systems experts may benefit from something like this and it might be really good for the community cause this might bring people in and growing their knowledge.

Developer C added:

The cross-linkages are really useful, the CVS logs, the bug reports that are related, all of that made perfect sense.

However, Dhruv may be less helpful to people who are already understand the source code of the community thoroughly and are well aware of the interactions in the community, as Developer B notes.

Potential and Scope

All participants saw good potential in a community semantic web, as embodied by Dhruv. They particularly liked how the semantic web could be built from the 'outside' as it were. Developer C explains:

This is great. It is about very smart cross-linkage. Taking the data and making it viewing it a different way. Site-wide search and knowledge management type systems, which took less informed approach to this. All that deeply built into the guts of the code. You've come from the outside and built this model and constructed the linkages in a totally unobtrusive way and not requiring deep internal support for these linkages.

So, this is definitely an advantage. You don't have to install much software, don't require much extra code. ... It might make a difference. A lot of it is just knowing where to go. Some people don't even know about cvs logs, so they don't even know that they can view the code for any given set of apis on the web, without having to dig into the file first.

Dhruv has potential to help newcomers to the community understand the software and the interactions between different components of the system. Developer B explains:

> *I think you sort of accomplished your goal by providing so many pathways to different information. ... I mean, if someone is curious and trying to learn a little bit more about a particular bug, you've provided many different pathways to code that's related so you've learned a little bit about different related procedures just by clicking around on those links. ... It might make it easier for people who don't have experience with the toolkit to get involved and learn some of the dependencies. Look at and get a sense of, what particular bug interacts or might interact with what part of the system. It would could expose some of the complexities that remain hidden behind the curtain.*

Beyond the process of bug resolution, a community semantic web like Dhruv may have potential to support discussions in professional community forums. As Developer A explains:

> *[Dhruv could perhaps be used for discussions] That's what I'm thinking as you're talking about it. There's .. I have a lot of fodder in [a medical] community with hundreds, if not thousands of users and hundreds and thousands, if not billions of messages, where there's content and there's images that are related to the text content. And people related to the specific areas within that group.*
>
> *That's one of my research interests in implementing something similar based on a existing ontology, where you can build up a KB with individuals within the community contributing back. Like photo.net. You have something similar to that, where you have individuals, intermediates showing the beginners and filtering stupid questions out from the experts. Cause you want to be able to support a large group of individuals. It's the same thing in this community. You know, you won't want [the most productive developer] getting all the requests. If there's something simple, you want to send it to newer people that just jumped on ... that are trying to build their respect within the community.*

> *And take advantage of that and at the same time, not taxing the experts that are actually the draw for new people. Yeah, totally cool.*

All the participants felt that the OpenACS community would welcome a system like Dhruv and be eager to try to deploy it in the community and observe its actual impact. As Developer C expressed:

> *I think there would be a lot of enthusiasm from the community to say that [the researcher] came up with some enhancements and it requires almost no code and we want to deploy it and see if you like it. Think about it.*
>
> *I like the fact that you didn't have to come and ask us to write a bunch of code, right? This is a gift basically and none of us had to do any work. Now we would have to do work to deploy it. But now that we can see some value, maybe it's worth it. I'd like for us to give it a spin and see what the impact is. Does it help people? Is it confusing?*

Developer C assessed the potential for deploying a system like Dhruv for OSS communities and explained what is involved. In the following, he refers to Dhruv's knowledge base as the 'model' and the 'engine' is Dhruv's procedure for generating cross-links.

> *To the extent that you want to donate what you've got to people, if you show them how easy it is to incorporate, make sure it's general, figure out (I can help you with this) an approach. This system requires a certain amount of human time to develop the model and that's non-negligible. Once you've got the model and the engine, you can make that available and teach people the techniques or volunteer your time if you're willing. So, you get the model and the self-contained engine that can potentially run anywhere. and then developing a set of processes for [deploying it] into live sites. Now, so the processes for incorporating it in terms of human time which is probably the most expensive resource, developing the model is probably the hardest part. The second part is kind of the training on how to incorporate the output. And the last part is just giving somebody a kind of box that just runs this and does*

> *the calculations once a night. 'Cause it could potentially run on somebody's desktop machine at home and then nightly copies up to the server. So that's cheap. So, the thing that the OSS communities will be most challenged on is they'll probably be able to find some resources to run this stuff. There are probably motivated coders in every community that once they see the potential they'll say wow we could put that into our system, let's do it. And then the hardest part is scaling up the whole adoption of the expertise of constructing the models and deciding how to handle that. That's going to be the biggest problem with adoption is developing the right model for each community. If you were to take some system that's widely deployed already. So, take one of those systems, develop a standard model and then any of the communities that run on that could potentially ... But the modeling is the hardest part to scale up. We're definitely happy to help you figure out the process of the middle part of that all, which is the how do we incorporate the results into an actual site in as low impact a way as possible. Very cool!*

The participants recognized the value of the Dhruv as a vehicle for demonstrating the value of a community semantic web. Developer C explained:

> *The thing I like most about this is honestly almost more of an intellectual thing, or an aesthetic thing. The notion that you're able to construct all this. Maybe this is what ... I have trouble with the term Semantic Web because it means a lot of things to a lot of people. but I understand something very clearly about what you did now that you've shown it to me. Which is that you've developed a model of the community, processes and so forth and you've applied some of the standard tools to that model and you've come up with something that's useful. or at least looks like it could be useful. Maybe you applied semantic ideas to content that happened to be on the Web ... I don't know if that makes it the semantic web .. but in any case, it's really useful, or at least it has the potential to be, it looks like it could be. I'd like to find out if it'd be.*

To summarize, the OpenACS community found the enhanced semantic interface presented by Dhruv to be very useful. They did find the highlighting of potential meaningful terms to be too low in precision, such that too many generic terms were highlighted. However, they did mention that these terms could easily be filtered out by a human. In addition, the cross-links pages present useful information, especially when the cross-links are small in number. As cross-links pages link to more artifacts, their usefulness declines sharply.

The message recommendations would found to be less directly useful for bug resolution. One participant suggested the recommendations would only be useful in the second or third pass of analysing the bug report, because it did not add directly to the information in the bug report itself. Others felt the message recommendations were generally a good idea, but it was not clear whether the recommended artifacts were relevant. This was partly due to the often misleading titles of bug reports and discussion threads. The community members did feel that this component would need improvement in order to be usable by the community.

The community also offered numerous suggestions for improvement of the Dhruv system and were generally very positive about its potential use within OpenACS and in OSS communities in general.

Chapter 5

Discussion and Future Work

In this chapter, we discuss the results of the evaluation presented in the previous chapter and interpret its implications. We first discuss the Dhruv system itself in section 5.1. Next, we focus on the various aspects of the Semantic Web as they were used in Dhruv in section 5.1.1. The potential of Dhruv to support OSS communities is discussed in section 5.1.2. Finally, we list the contributions of this work in section 5.3.

5.1 Dhruv

In Chapters 3 and 3.6, we presented a prototype of a community semantic web, Dhruv, for the OpenACS OSS community. Dhruv exploits an explicit representation of the semantic connections between community artifacts to present semantically relevant information to bug resolution participants. The evaluation in Chapter 4 revealed that Dhruv is often able to mimic part of the human processing of information and suggest artifacts earlier which would have been been used in the resolution of the bug ultimately. The similarity of recommendations for explicitly related bug reports indicates that Dhruv processing does reveal the same kinds of things as human processing. In addition, Dhruv can learn from previous links made explicitly by users to provide better recommendations, as shown in Section 4.3.4.

Inter-artifact links provide surprisingly useful information to Dhruv. Inter-artifact links are formed when the community explicitly links various artifacts together during the natural course of their interactions. These semantic cross-links are also rich enough to be useful in predicting the artifacts that are likely to be useful for a bug report. Inter-artifact links can be easily captured and represented by the Semantic Web beyond OSS communities, such as in discussion-oriented online communities.

Another point emphasized by the evaluation is that Dhruv can only attempt to find meaning in what already exists in a message. If the messages are poor in content and meaningful terms, the recommendations of Dhruv are also poor. Thus, when evaluating the recommendations for explicitly related bug reports, using the bug reports with richer text gives better recommendations than using terse bug reports.

The analysis of the recommendations indicates whether Dhruv can identify correct artifacts for a bug report. However, it does not inform us about whether the recommendations of Dhruv are meaningful. Since the objective of a system like Dhruv is to trigger the right thoughts among people resolving a bug report, it is important to evaluate the usefulness of the information presented by Dhruv separately. For this, we turned to the user study.

The user study reveals that people did find the information presented by Dhruv meaningful. Dhruv presented cross-links for code terms and noun phrases that were generally thought to be informative by OSS community members. Dhruv did tend to be relatively liberal in its selection of terms to expand on. These were however not considered to be distracting, as OSS community members could easily scan the message and focus only on useful highlighted terms, filtering the other less useful terms out.

An unexpected outcome of the user study was that community members felt they were unlikely to use the recommendations regardless of the quality of the recommendations. Instead they found Dhruv's enhanced semantic interface to the bug report messages to be far more valuable. As one of the participants commented, the enhanced message interface helped them better interpret the terms in the bug report message. In contrast, the message recommendations primarily provided secondary-level information of related artifacts for the entire bug report and were unlikely to help directly with fixing the bug itself. Thus, the enhanced message interface supports existing activities of people around the bug report, whereas the recommendations provide potentially useful information, but outside the context of current activities of people.

Quality may also have been a problem with the message recommendations. Several participants mentioned that they did not find the message recommendations too relevant. There are several possible explanations for this. One possibility is that the precision and recall achieved by the system is not adequate for the people examining the bug report. The recommendations need to be directly relevant to the bug report in order to be useful. Tangentially relevant recommendations are not good enough. However, the tangentially relevant recommendations may themselves be very useful for a newcomer to the community, as noted by a user study participant. Additionally, as one of the participants mentioned, the titles of the artifacts, especially in the case of bug reports and discussion threads, may not be representative of the content of those artifacts. People may therefore not be able to judge the relevance of a bug report from its title. More investigation is required to understand why people did not find the message recommendations to be relevant.

The fact that the OpenACS community members found the enhanced information interface more valuable than the message recommendations has wider implications for the Semantic Web vision within online community contexts. It suggests that exposing semantic information to people such that they can browse the semantic links between artifacts is likely to be meaningful to people, rather than merely using the semantic information to compute related artifacts. The strength of the Semantic Web is its power to provide explanations for why certain artifacts are related or recommended. Within collaborative work contexts, such explanations can be as useful as the final answer.

A key feature of the Dhruv community semantic web prototype is that it is built from the 'outside'. In other words, it represents a way to take an existing online OSS community and transform it into a semantic community. This contributes to the ease of deployment for the Semantic Web and reduces the adoption barrier for online communities.

The ontologies in Dhruv are fairly general and domain-independent. They can be used for a community semantic web by any OSS community built around an OpenACS website. Modeling other OSS communities will require some modification of Dhruv, primarily in the processes used to gather semantic metadata in OpenACS.

Various components of Dhruv, such as the metadata extraction and the heuristics for determining cross-links are independent modules that can be extended depending on needs. The recommendations of Dhruv can potentially be improved by plugging in state-of-the-art tools and techniques for these modules.

One possible extension of the cross-links heuristics is to exploit the sub class hierarchy of code concepts and to tie it more closely with the presented metadata. interesting thing to let each level of subclass provide its own incremental nested information in the cross-link heuristics; exploit subclass hierarchy and tie it more closely with presented information and cross-links

The current version of Dhruv represents a initial step in the realization of a comprehensive community semantic web. By modeling individual communities and community processes more closely and by developing more specific ontologies, we can improve on the current version to provide much more comprehensive and tailored support to online professional communities.

5.1.1 Use of the Semantic Web in Dhruv

Dhruv is an initial prototype of a community semantic web for OSS communities. The creation of Dhruv relied on light-weight processes that parsed existing web content and transformed it into semantic web content without interfering with the natural activities of the community. Making the transition from the Web to the Semantic Web as seamless as possible is an important requirement for Semantic Web applications.

There are several issues that the Semantic Web community needs to address before the Semantic Web can be widely used. To begin with, the community requires clearer modeling guidelines for ontology creation. For people who are not knowledge engineers, formal modeling is a difficult activity. It is not clear how to make modeling decisions, such as defining something as an individual or a class, or how finely to describe classes and what the effect of the description is on the reasoning performance.

A major obstacle in the creation of Dhruv is the large amount of data that is generated by the OSS community. The reasoners we tried to use within this work, namely Racer and Pellet, were unable to reason efficiently for large data. If the metadata input to the reasoners is inconsistent, then identifying which statement in the hundreds of statements caused an error to be flagged is itself an non-trivial task. This points to a huge gap for the Semantic Web. Reasoning about information on the Web necessarily brings with it the specter of huge data. Due to this, Dhruv can not take advantage of the full expressivity of description logics provided by OWL DL. Dhruv relied on cached information to generate its pages.

There is one system that does address the problem of efficient reasoning for large numbers of individuals: instance Store (iS) [HLTB04]. The iS system stores assertions about individuals and their types in a database, reducing reasoning over individuals to terminological reasoning. However, the current version of iS is limited to role-free reasoning of individuals, i.e. the ABox may have no axioms asserting role relationships between individuals. During the creation of Dhruv, this was deemed to be a major limitation. However, ultimately the primary use of ontologies in Dhruv is for the description, annotation and retrieval of large numbers of individuals. In hindsight, iS was probably the most appropriate system to use in Dhruv.

Dhruv does not make use of the open world assumption nor does it make use of ontologies distributed over multiple sites. As a small, self-contained community semantic web, Dhruv does not require them. It is simpler to implement Dhruv as a closed world with stable ontologies for individual communities. This also simplifies the ontology descriptions and reasoning. An ontology language like OWL then becomes primarily useful in linking up the ontologies of individual communities and enabling interoperation among them.

Within the current implementation of Dhruv, the inferencing and classification capabilities of the Semantic Web are underutilised. This is purely because current reasoners do not scale well to the numbers of objects dealt with by Dhruv. There are several possible uses for classification within Dhruv. A prime example is using inference to classify people into roles, such as bug fixer or core developer. The people recommendations can then use this role information to suggest appropriate people. For example, inactive members or people who have never participated in a bug resolution are unhelpful people recommendations for a bug report.

An interesting aspect of a community Semantic Web that was explored in the evaluation was its potential to change as a result of people using it. The Semantic Web can learn from how people use it. Frequently used links can be strengthened and infrequently used links de-emphasized. In the evaluation, we explored the effect of learnt links on the recommendations given by Dhruv. We found that learnt links increased the precision of Dhruv's recommendations. Thus Dhruv could use previously used links to give more relevant recommendations.

5.1.2 Supporting Open Source Software Communities

A community semantic web has the potential to provide enhanced support for activities in open source software communities and online professional communities, more generally.

In the case of Dhruv, we found different audiences within the community have different ways of using system. For example, some features of the system are relevant to newcomers to the community, such as the message recommendations for people, while other features, such as the highlighted terms with cross-links pages, are more useful to experts in the community.

Some of the participants in the user study noted the value of Dhruv for someone perusing bug reports after they have been fixed. By letting people find related artifacts for terms in a bug report, Dhruv places the bug report in the context of other artifacts in OpenACS. This can potentially help community members go back to major bugs and identify major trends or problem areas in the software source code.

Newcomers to the community can also benefit from seeing related artifacts and understanding how the bug report was affected by other artifacts and changes it effected in the software. The differences in the needs of newcomers and experts suggests that personalizing Dhruv for specific roles and tasks would make it more useful for the roles and tasks it serves. Thus, a Dhruv tailored for newcomers might emphasize the people recommendations and show message recommendations. The experts version of Dhruv might eschew the message recommendations altogether, focusing purely on the enhanced message interface.

The extent to which the community itself can take charge of the ontologies supporting their community will be a key determinant of the long-term use of a community semantic web. In order to suit their specific context and their changing needs, the community needs to be able to modify Dhruv and extend it incrementally. In particular, the ontologies are often thought to be difficult to modify for people who are not trained in knowledge representation. However, an OSS community may not need complex and comprehensive ontologies to benefit from semantic modeling of the community. In Dhruv, the ontologies used are relatively simple and yet useful. Thus, extending the ontology may be of comparable difficulty to extending a database schema. In addition, in Dhruv, the ontological concepts and the processing of messages are closely tied together. Thus, it is difficult to add additional concepts without also adding additional processing capabilities, such as parsing and identifying cross-links, for Dhruv. It is also possible that the community will evolve a special role for the maintenance of the ontology and the related processing capabilities.

Generalizing Dhruv to other OSS communities and online professional communities requires understanding what the semantics of the content is and how it is manifested in community interactions. For other OSS communities, the semantics of the content is likely to be same as in OpenACS, namely references to code objects, bug reports etc. as well as community jargon. The only difference may lie in the programming languages used, which would affect the information extraction rules in the metadata-extraction component of Dhruv. The next step is to develop appropriate ontologies that can be used to give machine-comprehensible semantics to the terms used in interactions. For OSS communities, Dhruv already provides a set of ontologies that are fairly general. Across OSS communities, Dhruv should be fairly simple to generalize. Dhruv is independent of the programming language and programming domain used. It is only the source code architecture that differs across communities and that may require some modifications of Dhruvs code ontology. However, Dhruvs code ontology itself is primarily an upper ontology for software code. The only OpenACS-specific component of the ontologies are a small number of concepts and relations (less than ten) in the code ontology. Another OSS community wanting to use Dhruv will need to remove these concepts and relations and possibly add their own community-specific concepts. The other ontologies in Dhruv are domain-independent. The heuristics for identifying cross-links are based primarily on the ontologies, so modifications to the heuristics depend on the extent of modification of Dhruv ontologies. The recommendation procedures used by Dhruv are domain-independent for OSS communities.

During the course of this work, several requirements of the automatic processing of OSS content were identified:

- Always acknowledge the person making a contribution, in particular patches to the source code.
- Maintain unique identities of people. When community members keep different identities, it becomes more difficult to track the activities of each member. This reduces the accuracy of people recommendations.
- Keep canonical form of files in patch diffs. The patch diffs contain the file names and paths of files that are modified by the patch. If the developer creating the patch does not use canonical file paths, the paths recorded in the patch diffs will not be correct. Dhruv would then need additional processing to identify the most probable file being referred to.
- Maintain unique identities of software objects. Refer to software objects, particularly those in namespaces consistently and with full namespaces. This reduced processing burden on Dhruv to attempt to identify the right software objects.

Dhruv currently does not allow community members to explicitly search for cross-links for a given term. Some of the participants mentioned this as a worthwhile addition to Dhruv. Landsdale [Lan88] showed that every attempt to retrieve information in personal information management is based on two different psychological processes: recall-directed search followed by recognition-based scanning. Currently, Dhruv relies on a bug report message to serve as the input for the recall-directed search. However, it is relatively simple for Dhruv to support searching for cross-links for terms people already know are relevant.

In contrast to other groupware systems, such as the Coordinator and Answer Garden, Dhruv follows a non-interventionist approach, working in the background as far as possible without interrupting the existing workflow. Dhruv observes the ongoing human activities and tries to intelligently interpret the communication exchange and provide its own input in the process. Dhruv does not suffer from some of the problems of previous CSCW systems, because OSS communities are naturally more reliant on online, electronic communication.

An additional feature of Dhruv is that it can be constructed more or less automatically from the community space, capturing the core semantics of community interactions, without requiring the community members to change their way of working. In the future, Dhruv can be generalized to support the interactions in any online professional community.

5.2 Lessons Learnt

The most important lesson learnt as a result of this work is that the Semantic Web needs ways to handle large amounts of data. The Semantic Web reasoning infrastructure will need to deal with significant amounts of data on the Web and current reasoners for OWL are easily overwhelmed with data. For this reason, we were not able to make full use of the semantic inferencing possibilities offered by the Semantic Web. Instead, we opted for a simpler, less expressive set of ontologies for modeling the OpenACS community. The fact that this simpler modeling also proved to be so valuable to the OpenACS community attests to the value of semantics and the lack thereof in current systems.

The Semantic Web also needs more domain-specific ontologies. Although there has been an effort to develop upper ontologies for several domains, these need to be linked to more specific ontologies or categories that are used within actual work domains. Thus, for Dhruv, we could not make use of any existing ontology. Instead, we needed to construct specialized ontologies for use in the context of the community work artifacts.

The user study indicated that OpenACS community members were keen on being involved in the improvement and extension of Dhruv. The community is technically sophisticated enough to understand how Dhruv works and to handle and maintain the explicit semantics used within Dhruv. This is an audience unlike that of typical ontology-based systems, where there is a sharp distinction between the users of the system and the knowledge engineers who design the system. Furthermore, systems that do not require developers to change their system or ways of working are well-regarded. The community appreciated the lightweight and non-intrusive nature of Dhruv in comparison to other knowledge management systems, citing it as a major factor in their enthusiasm for Dhruv.

5.3 Contributions

The primary contribution of this research work has been to demonstrate that capturing the semantics of a comprehensive set of artifacts can support bug resolution in OSS communities. In particular, we developed:

1. A model and knowledge base for the OpenACS community, their software content and interactions. The model can be generalized to any OSS community and to other online professional communities.

2. Tools and techniques to automatically extract and annotate Web information with respect to the ontologies mentioned above. In addition, we developed rules and heuristics to generate cross-links between OSS artifacts.

3. A framework for the use of the Semantic Web in the context of OSS development and more generally, for collaboration in web communities as well as in software development. We demonstrated the potential of a community Semantic Web for OSS communities.

With respect to research in the Semantic Web community, our work is the first to focus on supporting problem-solving in Web communities. Given that thriving Web communities have been integral to the success of the Web, it is imperative that the evolution of Semantic Web communities from Web communities be explored. However, this area of research has remain essentially unexplored until now. In this work, we demonstrated the transition of the content of an existing Web community, OpenACS, to Semantic Web content. In addition, we explored how the Semantic Web can support problem-solving interactions within Web communities by providing supporting information from the existing archive of the community.

Another contribution of this research in the Semantic Web area is to develop several domain-specific ontologies to model OSS communities. The ontologies describe various interaction artifacts of OSS development, such as bug reports, discussion threads and commit log information, and associated web community processes. The code ontology describes OpenACS code and can be used with slight modifications for other OSS community websites.

With respect to OSS communities, we have demonstrated that a hybrid approach combining information from various community artifacts and combining various kinds of techniques and heuristics can support bug resolution. We identified how the semantics inherent in community interactions can be semi-automatically processed and used to support the community. We found that individual information sources and individual techniques did not provide as useful information as their combination did. Through the prototype Dhruv, we demonstrated the potential use of the Semantic Web in OSS communities.

5.4 Future Work

There are several possible future directions for this work. The most promising direction of future work is actual deployment of Dhruv in the OpenACS community. The user study revealed that the community is highly enthusiastic about Dhruv and ready to help to bring about the eventual use of Dhruv in the community. Deploying Dhruv in the OpenACS community will allow the community to get involved in the use, maintenance and future evolution of Dhruv.

Real-world deployment will also give us the opportunity to improve individual components of Dhruv. In this work, we focused on simple techniques to demonstrate a proof-of-concept. To take the prototype and transform it into a working system, we need to use more sophisticated state-of-the-art techniques for various components of Dhruv, such as metadata extraction and the generation of cross-links.

The user study revealed that the message recommendations provided by Dhruv were not directly useful to the bug resolution task. The feedback from the study participants suggested that the message recommendations were not the right kind of support for bug resolution, at least in the first pass through the bug report. This is an interesting finding, since recommendation systems are a successful genre of systems in their own right. Perhaps there are some subtleties in the task of bug resolution that make recommendations unsuitable support during bug resolution. It may be that the recommendation systems provide information that is more peripheral than central to the task of bug resolution. Future work should identify how Dhruv recommendations can be made more useful to the OpenACS community, either by improving the recommendations themselves or by identifying alternate forms of recommendation support that are useful for work contexts.

It has also been suggested by the study participants that Dhruv is likely to be particularly useful for novice developers, who wish to participate more substantially in the community. By making it easier for newcomers to understand the context of a bug report and to explore the source code related to the bug report, Dhruv helps newcomers to participate in bug resolution more effectively and perhaps even develop fixes for the bug. This is likely to help attract new developers to the community and help compensate for the typical lack of documentation in OSS communities that turns away novice developers from a community and a code base. Future work to extend Dhruv to support newcomers more thoroughly and explicitly can have significant effect on OSS communities.

The concept underlying Dhruv is fairly simple and general: identify a structured portion of the semantics of interactions and attempt to support interactions by making the semantics explicit. There is huge scope for applying this concept to other contexts beyond bug resolution in OSS communities. The enhanced semantic interface provided by Dhruv is likely to be directly useful in the task of code comprehension, which is a pre-requisite for bug resolution. The task of trying to understand the code is less tightly focused than bug resolution and involves a high degree of exploration of the links between software objects. Dhruvs enhanced semantic interface supports precisely this type of exploration and is therefore likely to be useful for code comprehension too.

Beyond OSS communities, there are other online professional communities that have a core of relatively structured content which is high in semantics. Obvious examples are educational communities and communities conducting scientific research. Both types of communities are likely to benefit from a system that supports current interactions in the community by making the interaction history of the community more transparent. By extending Dhruv to support both these types of communities, we can demonstrate the generality of the concept underlying Dhruv as well as make Dhruv itself more domain-independent.

Bibliography

[AB02] U. Asklund and L. Bendix. A study of configuration management in open source software projects. *IEEE Proceedings on Software*, 149(1):40–46, February 2002. 2.1.2, 2.1.4, 1, 2.1.5

[Apa] Apache software foundation. http://www.apache.org/. 2.1

[BCM+03] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003. 2.3.2, 2.3.2, 2.3.2, 2.3.2

[BDH+05] Victoria Bellotti, Nicolas Ducheneaut, Mark Howard, Ian Smith, and Rebecca E. Grinter. Quality versus quantity: E-mail-centric task management and its relation with overload. *Human-Computer Interaction*, 20:89–138, 2005. 2.1.5

[BDS+90] Ronald J. Brachman, Premkumar Devanbu, Peter G. Selfridge, David Belanger, and Yun Chen. Toward a software information system. *AT&T Technical Journal*, 69(2):22–41, 1990. 2.2.2

[Bec04] Dave Beckett. Turtle–terse rdf triple language. http://www.ilrt.bris.ac.uk/discovery/2004/01/turtle/, January 2004. 4

[Ber93] Lucy M. Berlin. Beyond program understanding: A look at programming expertise in industry. In Jean C. Scholtz Curtis R. Cook and James C. Spohrer, editors, *Proceedings of the Fifth Workshop on Empirical Studies of Programmers*, pages 6–25, Palo Alto, CA, USA, December 1993. Ablex Publishing Corporation. 2.1.3

[BHS02] Bettina Berendt, Andreas Hotho, and Gerd Stumme. Towards semantic web mining. In Ian Horrocks and James Hendler, editors, *Proceedings of the*

First International Semantic Web Conference (ISWC), volume 2342 of *Lecture Notes for Computer Science*, pages 264–278, Sardinia, Italy, June 2002. Springer Verlag. 2.3.1

[BHS05] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics as ontology languages for the semantic web. In Dieter Hutter and Werner Stephan, editors, *Mechanizing Mathematical Reasoning: Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday*, volume 2605 of *Lecture Notes in Computer Science*, pages 228–248. Springer Verlag, January 2005. 2.3.2

[BJ04] Glenda Browne and Jonathan Jermey. *Website Indexing: enhancing access to information within websites*. Auslib Press, 2nd edition, 2004. 2.3.2

[BL98] Tim Berners-Lee. What the semantic web can represent, September 1998. 2.3

[BL02] Tim Berners-Lee. Relational databases and the semantic web. http://www.w3.org/DesignIssues/RDB-RDF.html, March 2002. 2.3.2

[BLCS99] Tim Berners-Lee, Dan Connolly, and Ralph R. Swick. Web architecture: Describing and exchanging data. http://www.w3.org/1999/04/WebData, June 1999. 2.3

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, May 2001. 1, 2.3, 2.3.2

[Bloa] http://www.blogger.com. 2.3.1

[Blob] http://www.blosxom.com. 2.3.1

[BM04] Dan Brickley and Libby Miller. FOAF vocabulary specification. http://xmlns.com/foaf/0.1/, May 2004. 2.3.1, 3.2.4

[BPS94] Alexander Borgida and Peter F. Patel-Schneider. A semantics and complete algorithm for subsumption in the CLASSIC description logic. *Journal of Artificial Intelligence Research*, 1:277–308, 1994. 2.3.2

[Bro83] R. Brooks. Toward a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies*, 18:543–554, 1983. 2.2.2

[Buga] OpenACS Bug Tracker. http://openacs.org/bugtracker/openacs/. 2.1.4

[Bugb] The bugzilla bug tracking system. http://bugzilla.mozilla.org/. 2.1.3

[Cas] Frederico Casalegno. Living memory - the role of memory in the communities. http://www.memoire-vivante.org/. 2.3.3

[Cas00] Frederico Casalegno. Living memory: une approche écologique de la mémoire en réseau. *Sociétés: Revue des sciences humaines et sociales*, No. 68, February 2000. 2.3.3

[CDF$^+$00] M. Craven, D. DiPasquo, Dayne Freitag, Andrew McCallum, Tom Mitchell, Kamal Nigam, and Sean Slattery. Learning to construct knowledge bases from the world wide web. *Artificial Intelligence*, 118(1-2):69–113, 2000. 2.3.1

[CM03] Davor Cubranic and Gail C. Murphy. Hipikat: Recommending pertinent software development artifacts. In *Proceedings, International Conference on Software Engineering*, pages 408–418, Portland, OR, May 2003. 2.3.3

[Cro] Cross-link. http://en.wikipedia.org/wiki/Cross-link. 2

[CSS] Cascading style sheets. http://www.w3.org/Style/CSS/. 2.3.1

[Cvs] Concurrent Versions System (CVS). http://www.cvshome.org/. 2.1.3, 2.1.4

[DB92] Paul Dourish and Victoria Bellotti. Awareness and coordination in shared workspaces. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW)*, pages 107–114. ACM Press, 1992. 2.1.3

[DB03] Nicolas Ducheneaut and Victoria Bellotti. Ceci n'est pas un objet? talking about objects in e-mail. *Human-Computer Interaction*, 18:85–110, 2003. 2.1.4

[DB04] W3C Recommendation Dave Beckett, Editor. Rdf/xml syntax specification (revised). http://www.w3.org/TR/rdf-syntax-grammar/, February 2004. 4

[dBLPF05] Jos de Bruijn, Rubén Lara, Axel Polleres, and Dieter Fensel. OWL DL vs. OWL flight: conceptual modeling and reasoning for the semantic web. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 623–632, New York, NY, USA, 2005. ACM Press. 2.3.2

[DCvH$^+$02] Mike Dean, Dan Connolly, Frank van Harmelen, James Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. Web Ontology Language (OWL) Reference Version 1.0. OWL Specification, November 2002. 2.3.2

[DDM03] Martin Dzbor, John Domingue, and Enrico Motta. Magpie – Towards a semantic web browser. In Katia Sycara Dieter Fensel and John Mylopoulos, editors, *Proceedings of the Second International Semantic Web Conference (ISWC)*, volume 2870 of *Lecture Notes in Computer Science*, pages 738–753, Sanibel Island, FL, USA, October 2003. Springer Verlag. 2.3.3

[DGMN02] Jamie Dinkelacker, Pankaj K. Garg, Rob Miller, and Dean Nelson. Progressive open source. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 177–184, Orlando, Florida, USA, May 2002. ACM Press. 2.1.2

[Dot] dotLRN. http://openacs.org/projects/dotlrn/. 2.4

[Dub04] Dublin core metadata element set, version 1.1: Reference description. http://dublincore.org/documents/dces/, December 2004. 2.3.1

[DW05] Nicolas Ducheneaut and Leon A. Watts. In search of coherence: a review of e-mail research. *Human-Computer Interaction*, 20:11–48, 2005. 2.1.5

[Ecl] Eclipse.org. http://www.eclipse.org. 2.3.3

[Fir] The firefox web browser. http://firefox.mozilla.org/. 2.1

[Flo] http://flora.sourceforge.net/. 2.3.2

[FOAa] http://www.ldodds.com/foaf/foaf-o-matic.html. 2.3.1

[FOAb] http://www.formsplayer.com/demo/foaf/foaf-creator.html. 2.3.1

[FOAc] FOAFnaut. http://www.foafnaut.org/. 2.3.1

[FvHH$^+$01] Dieter Fensel, Frank van Harmelen, Ian Horrocks, Deborah McGuinness, and Peter F. Patel-Schneider. OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45, 2001. 2.3.2

[GC91] Edward M. Gellenbeck and Curtis R. Cook. An investigation of procedure and variable names as beacons during program comprehension. In Thomas G. Moher Jürgen Koenemann-Belliveau and Scott P. Robertson, editors, *Proceedings of the Fourth Workshop on Empirical Studies of Programmers*, pages 65–79, New Brunswick, NJ, USA, December 1991. Ablex Publishing Corporation. 2.2.2

[GDB04] J. Grant and W3C Recommendation D. Beckett, Editors. N-triples section in rdf test cases. http://www.w3.org/TR/rdf-testcases/, February 2004. 4

[GG02] Carl Gutwin and Saul Greenberg. A descriptive framework of workspace awareness for real-time groupware. *Journal of Computer-Supported Cooperative Work (JCSCW)*, 3-4:411–446, 2002. 2.1.3

[Gho03] Rishab A. Ghosh. Understanding free software developers: Findings from the FLOSS study. Working paper, MERIT/Institute of Infonomics, University of Maastricht, June 2003. 2.1.2, 4

[GHVD03] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: combining logic programs with description logic. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 48–57, New York, NY, USA, 2003. ACM Press. 2.3.2

[GO] Gene ontology. http://nciterms.nci.nih.gov/NCIBrowser/Connect.do?dictionary=GO. 2.3.2

[GPS04] Carl Gutwin, Reagan Penner, and Kevin Schneider. Group awareness in distributed software development. In *Proceedings of the Computer Supported Cooperative Work (CSCW)*, Chicago, Illinois, USA, November 2004. 2.1.3, 2.1.3, 2.1.3, 5, 2.1.5

[gre] http://www.greenpeace.org/international/footer/software-copyright. 2.4

[Gru93] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. Technical report, KSL, 1993. 2.3.2

[HH97] B. Heckel and B. Hamann. A visual e-mail analysis tool. In *Proceedings of the NPIV 97 Workshop on New Paradigms in Information Visualization and Manipulation*, New York, USA, 1997. ACM Press. 2.1.5

[HLTB04] Ian Horrocks, Lei Li, Daniele Turi, and Sean Bechhofer. The instance store: DL reasoning with large numbers of individuals. In *Proceedings of the 2004 Description Logic Workshop (DL 2004)*, pages 31–40, 2004. 5.1.1

[HM00] James Hendler and Deborah L. McGuinness. The DARPA agent markup language. *IEEE Intelligent Systems*, 15(6):67–73, November 2000. 2.3.2

[HM01] Volker Haarslev and Ralf Möller. Racer system description. In *International Joint Conference on Automated Reasoning (IJCAR)*, Siena, Italy, June 18-23 2001. 2.3.2

[Hor98] Ian Horrocks. Using an expressive description logic: FaCT or fiction? In *Proceedings of KR 1998*, 1998. 2.3.2

[HP02] Stefan Haustein and Jörg Pleumann. Is participation in the semantic web too difficult? In Ian Horrocks and James Hendler, editors, *Proceedings of the First International Semantic Web Conference (ISWC)*, volume 2342 of *Lecture Notes for Computer Science*, pages 448–453, Sardinia, Italy, June 2002. Springer Verlag. 2.3.1

[HPS03] Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In Katia Sycara Dieter Fensel and John Mylopoulos, editors, *Proceedings of the Second International Semantic Web Conference (ISWC)*, volume 2870 of *Lecture Notes in Computer Science*, pages 17–29, Sanibel Island, FL, USA, October 2003. Springer Verlag. 2.3.2, 2.3.2

[HPS04] Ian Horrocks and Peter F. Patel-Schneider. A proposal for an OWL rules language. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 723–731, New York, NY, USA, 2004. ACM Press. 2.3.2

[HPSB$^+$03] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A semantic web rule language combining OWL and RuleML. http://www.daml.org/2003/11/swrl/, 2003. 2.3.2

[HS02] Timothy J. Halloran and William L. Scherlis. High quality and open source software practices. In *Meeting Challenges and Surviving Success: 2nd Workshop on Open Source Software Engineering, International Conference on Software Engineering*, Orlando, FL, May 2002. 1, 2.1.2, 2.1.3, 2.1.3, 2.1.3, 2.1.4

[HvHPS01] Ian Horrocks, Frank van Harmelen, and Peter Patel-Schneider. The DAML+OIL Web Ontology Language. DAML Specification, March 2001. 2.3.2

[IBM] Ibm-apache. http://www.technewsworld.com/story/35527.html. 2.1.2

[Jen] The Jena semantic web framework. http://jena.sourceforge.net/. 2.3.2, 2.3.2

[KA86] Claudius M. Kessler and John R. Anderson. A model of novice debugging in LISP. In Elliot Soloway and Sitharama Iyengar, editors, *Proceedings of the First Workshop on Empirical Studies of Programmers*, pages 198–212, Washington, DC, USA, June 1986. Ablex Publishing Corporation. 2.2.2

[KLW95] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, May 1995. 2.3.2

[KPH05] Aditya Kalyanpur, Bijan Parsia, and James Hendler. A tool for working with web ontologies. *International Journal on Semantic Web and Information Systems*, 1(1):36–49, January-March 2005. 2.3.2

[KPO^{+}03] Atanas Kiryakov, Borislav Popov, Damyan Ognyanoff, Dimitar Manov, Angel Kirilov, and Miroslav Goranov. Semantic annotation, indexing and retrieval. In Katia Sycara Dieter Fensel and John Mylopoulos, editors, *Proceedings of the Second International Semantic Web Conference (ISWC)*, volume 2870 of *Lecture Notes in Computer Science*, pages 484–499, Sanibel Island, FL, USA, October 2003. Springer Verlag. 2.3.1, 3.6.1

[KR03] Joseph B. Kopena and William C. Regli. DAMLJessKB: A tool for reasoning with the semantic web. In *Proceedings of the Second International Semantic Web Conference (ISWC)*, Sanibel Island, FL, October 2003. 2.3.2

[KS95] Robert Kraut and L. Streeter. Coordination in software development. *Communications of the ACM*, pages 69–81, 1995. 2.1

[Lan88] M. Landsdale. The psychology of personal information management. *Applied Ergonomics*, 19:55–66, 1988. 1, 2.1.5, 5.1.2

[Let86] Stanley Letovsky. Cognitive processes in program comprehension. In Elliot Soloway and Sitharama Iyengar, editors, *Proceedings of the First Workshop on Empirical Studies of Programmers*, pages 58–79, Washington, DC, USA, June 1986. Ablex Publishing Corporation. 2.2.2, 3.6.1

[Lew82] C. Lewis. Using the 'thinking-aloud' method in cognitive interface design. Technical Report Research Report RC9265, IBM T. J. Watson Research Center, Yorktown Heights, NY, USA, 1982. 3

[Lin] The linux operating system. http://linux.org/. 2.1, 2.1.1

[LS99] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. RDF Specification, Feb 1999. 2.3.1

[LTT$^+$03] Yang Li, Simon Thompson, Zhu Tan, Nick Giles, and Hamid Gharib. Beyond ontology construction: Ontology services as online knowledge sharing communities. In Katia Sycara Dieter Fensel and John Mylopoulos, editors, *Proceedings of the Second International Semantic Web Conference (ISWC)*, volume 2870 of *Lecture Notes in Computer Science*, pages 469–483, Sanibel Island, FL, USA, October 2003. Springer Verlag. 2.3.3

[LUM$^+$02] Gangmin Li, Victoria Uren, Enrico Motta, Simon Buckingham Shum, and John Domingue. Claimaker: Weaving a semantic web of research papers. In Ian Horrocks and James Hendler, editors, *Proceedings of the First International Semantic Web Conference (ISWC)*, volume 2342 of *Lecture Notes for Computer Science*, pages 436–441, Sardinia, Italy, June 2002. Springer Verlag. 2.3.3

[McB02] Brian McBride. Four steps towards the widespread adoption of the semantic web. In Ian Horrocks and James Hendler, editors, *Proceedings of the First International Semantic Web Conference (ISWC)*, volume 2342 of *Lecture Notes for Computer Science*, pages 419–422, Sardinia, Italy, June 2002. Springer Verlag. 2.3.3

[MEG$^+$03] Luke McDowell, Oren Etzioni, Steven Gribble, Alon Halevy, Henry Levy, William Pentney, Deepak Verma, and Stani Vlasseva. Mangrove: Enticing ordinary people onto the semantic web via instant gratification. In Katia Sycara Dieter Fensel and John Mylopoulos, editors, *Proceedings of the Second International Semantic Web Conference (ISWC)*, volume 2870 of *Lecture Notes in Computer Science*, pages 754–770, Sanibel Island, FL, USA, October 2003. Springer Verlag. 2.3.3

[MFH00] Audris Mockus, Roy T. Fielding, and James Herbsleb. A case study of open source software development: the apache server. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 263–272, New York, NY, USA, 2000. ACM Press. 1

[MFH02] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3), July 2002. 2.1.3, 2.1.3

[Mil00] David R. Millen. Community portals and collective goods: Conversation archives as an information resource. In *Proceedings of the 33rd Annual Hawaii International Conference on Systems Sciences (HICSS)*, Maui, Hawaii, USA, January 4-7 2000. 2.1, 2.1.5

[Mov] http://www.movabletype.org. 2.3.1

[Moz] The mozilla software. http://mozilla.org/. 2.1, 2.1.1

[MS86] F. Warren McFarlan and Donna B. Stoddard. *Otisline*. Harvard Business School Case, June 1986. 2.3.3

[MvH04] Deborah L. McGuinness and Frank van Harmelen. OWL web ontology language overview. http://www.w3.org/TR/owl-features/, February 2004. 2.3.2

[MYR03] Saikat Mukherjee, Guizhen Yang, and I. V. Ramakrishnan. Automatic annotation of content-rich HTML documents. In Katia Sycara Dieter Fensel and John Mylopoulos, editors, *Proceedings of the First International Semantic Web Conference (ISWC)*, volume 2870 of *Lecture Notes in Computer Science*, pages 533–549, Sanibel Island, FL, USA, October 2003. Springer Verlag. 2.3.1

[NCI] NCI thesaurus. http://nciterms.nci.nih.gov/NCIBrowser/. 2.3.2

[Nie93] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann, 1993. 3

[NSC$^+$01] N. F. Noy, M. Sintek, M. Crubezy, R. W. Fergerson, and M. A. Musen. Creating semantic web contents with protege-2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001. 2.3.2

[Ope] OpenACS: Open architecture community system. http://openacs.org/. 2.1.1, 2.1.2, 2.1.3, 2.2.1, 2.4

[OTH$^+$04] Ikki Ohmukai, Hideaki Takeda, Masahiro Hamasaki, Kosuke Numa, and Shin Adachi. Metadata-driven personal knowledge publishing. In Sheila McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *Proceedings of the Third International Semantic Web Conference (ISWC)*, volume 3298 of *Lecture Notes in Computer Science*, pages 591–604, Hiroshima, Japan, November 2004. Springer Verlag. 2.3.1

[OWL] OWL API. http://sourceforge.net/projects/owlapi. 2.3.2

[Pho] Photo.net. http://www.photo.net/. 1

[Pil02] Mark Pilgrim. What is RSS? http://www.xml.com/pub/a/2002/12/18/dive-into-xml.html, December 2002. 5

[Pla] PlanetPDF. Planetpdf. http://www.planetpdf.com/. 2.1

[QHK03] Dennis Quan, David Huynh, and David R. Karger. Haystack: A platform for authoring end user semantic web applications. In Katia Sycara Dieter Fensel and John Mylopoulos, editors, *Proceedings of the Second International Semantic Web Conference (ISWC)*, volume 2870 of *Lecture Notes in Computer Science*, pages 738–753, Sanibel Island, FL, USA, October 2003. Springer Verlag. 2.3.3

[R] The R statistical software. http://www.r-project.org/. 2.1.1

[RSS] RDF site summary (RSS) 1.0. http://web.resource.org/rss/1.0/. 2.3.1

[Sel90] P. Selfridge. Integrating code knowledge with a software information system. In *Proceedings of Fifth Conference on Knowledge-based Software Assistance*, pages 183–195. IEEE Press, September 1990. 1

[Sem] W3C semantic web activity. http://www.w3.org/2001/sw/. 2.3

[SL86] Elliot Soloway and Stan Letovsky. Delocalized plans and program comprehension. *IEEE Software*, 3(3), 1986. 2.2.2

[SLPL86] Elliot Soloway, Stan Letovsky, Juan Pinto, and Diane Littman. Mental models and software maintainence. In *Proceedings of the Conference on Empirical Studies of Programmers*, pages 80–98. Ablex Publishers, 1986. 2.2.2

[Smi02] Marc Smith. Tools for navigating large social cyberspaces. *Communications of the ACM*, 45(4):51–55, 2002. 2.1.3, 2.1.5

[SSO87] Elliot Soloway, S. Sheppard, and Gary Olson, editors. *Proceedings of the Second Workshop on Empirical Studies of Programmers*. Ablex Publishers, December 1987. 2.2.2

[SSS91] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991. 2.3.2

[SWM04] Michael K. Smith, Chris Welty, and Deborah L. McGuinness. OWL web ontology language guide. http://www.w3.org/TR/owl-guide/, February 2004. 2.3.2

[TSL93] L. Terveen, P. Selfridge, and M. Long. From folklore to living design memory. In *Proceedings of INTERCHI'93*, pages 15–22, 1993. 2.1.3

[Ubu] Ubuntu. http://www.ubuntulinux.org/. 2.1

[URI] Uniform resource identifier (uri) activity statement. http://www.w3.org/Addressing/Activity. 2.3.1

[Val] OWL validator. http://owl.bbn.com/validator/. 2.3.2

[vKSL03] Georg von Krogh, Sebastian Spaeth, and Karim Lakhani. Community, joining and specialisation in open source software innovation: A case study. Working paper, MIT Sloan School of Management, June 2003. 2.1.2

[w3c] World wide web consortium. http://www.w3.org/. 2.3

[Web] http://en.wikipedia.org/wiki/Weblog. 6

[Wel95] Christopher A. Welty. *An Integrated Representation for Software Development and Discovery*. PhD thesis, Rensselaer Polytechnic Institute, 1995. 1, 4, 2.1.5, 2.2.2

[Wel97] Christopher Welty. Augmenting abstract syntax trees for program understanding. In *Proceedings of the 1997 Automated Software Engineering Conference*. IEEE Computer Society Press, 1997. 2.2.2

[Wie86] S. Wiedenbeck. Beacons in computer program comprehension. *International Journal of Man-Machine Studies*, 25:697–709, 1986. 2.2.2

[Wik] Wikipedia. http://wikipedia.org. 1, 2.1

[XEm] The xemacs editor. http://xemacs.org/. 2.1.1

[XFr] Xfree86. http://www.xfree86.org/. 2.1.1

[Zub88] Shoshana Zuboff. *In the Age of the Smart Machine: The Future of Work and Power*. Basic Books Inc., 1988. 2.2.1