Uncertainty in Self-Adaptive Systems

Categories, Management, and Perspectives

**Javier Cámara, David Garlan, Won Gu Kang,
Wenxin Peng, and Bradley Schmerl**

July 2017
CMU-ISR-17-110

Institute for Software Research
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

 This material is based upon work funded and supported by the Department of Homeland Security under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense, the National Security Agency, and AFRL and DARPA under agreement FA8750-16-2-0042. The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation. This report was prepared for the SEI Administrative Agent AFLCMC/AZS 5 Eglin Street Hanscom AFB, MA 01731-2100 NO WARRANTY.

Keywords: uncertainty, self-adaptation, decision-making, formal reasoning

Abstract

Self-Adaptive systems are expected to adapt to unanticipated run-time events using imperfect information about their environment. This entails handling the effects of uncertainties in decision-making, which are not always considered as a first-class concern. This technical report summarizes a set of existing techniques and insights into addressing uncertainty in self-adaptive systems and outlines a future research agenda on uncertainty management in self-adaptive systems. The material in this report is strongly informed by our own research in the area, and is therefore not necessarily representative of other works.

Contents

Chapter 1

Introduction

Complex software-intensive systems are increasingly relied upon in our society to support tasks in different contexts that are typically characterized by a high degree of *uncertainty*. Self-adaptation [**?**, **?**] is regarded as a promising way to engineer in an effective manner systems that are *resilient* to run time changes in spite of the different uncertainties in their execution environment (e.g., resource availability, interaction with human actors), goals, or even in the system itself (e.g., faults and inaccuracies in sensors).

During the last few years, the research community has made an important effort in supporting the analysis and management of self-adaptive systems under uncertainty. However, although different studies have dealt with the classification of the different dimensions and sources of uncertainty [EM13, MHAW16, **?**], representation and formal reasoning about uncertainties has been done in a rather ad-hoc manner in works on self-adaptation.

This technical report summarizes a set of existing techniques and insights into addressing uncertainty in self-adaptive systems, as well as outlining a future research agenda on uncertainty management in self-adaptive systems.[1]

The report starts by summarizing background material in Chapter 2, including a categorization of the different dimensions of uncertainty, followed with theories and techniques employed for managing different types of uncertainties in self-adaptive systems, and finishing up with a summary of techniques for managing uncertainty in related areas like artificial intelligence or cyber-physical systems.

The central part of the report builds on the material presented in Chapter 2, going into more depth about how to represent different uncertainties in self-adaptive systems, narrowing down the scope to MAPE-K systems. Concretely, Chapter 3 identifies types of uncertainties in different parts of MAPE-K systems, and uses the Rainbow framework for architecture-based self-adaptation, an instance of MAPE-K, to illustrate how different types of uncertainties can be captured in architecture-centric models.

In Chapter 4, we show how to use models of uncertainty about the accuracy of sensor information as an example of how we can explicitly reason about uncertainty in decision-making for adaptation. We show that incorporating an explicit representation of uncertainty into reasoning can result in better performance of the adaptive system.

[1] The material in this report is strongly informed by our own research in the area, and is therefore not necessarily representative of other works.

Chapter 2

Background and Related Work

During the last few years, the research community has made an important effort in supporting the construction of self-adaptive systems. In particular, approaches to identify the added value of alternative design decisions have explored different theories (e.g., probability [Kol56], fuzzy sets and possibility [Zad65, Zad99], or games [Mye91]) to factor in and mitigate the various types of uncertainty that affect self-adaptive systems.

Moreover, recent advances in formal verification [CFK+13, KNP09] have also explored the combination of probability and game theory to analyze systems in which uncertainty and competitive behavior are first-order elements.

This chapter first overviews the different dimensions of uncertainty according to the classification provided in [MHAW16]. The remainder of the chapter discusses how different theories have been employed to analyze self-adaptive systems under uncertainty. We categorize the different approaches according to the theories employed and the different sources of uncertainty that they target conforming to the classification provided by Esfahani and Malek in [EM13] (Table 2.3).

2.1 Dimensions of Uncertainty

There have been several papers on identifying types of uncertainty [EM13, WSB+09, CSBW09], but in this overview we will focus on the study by Sara Mahdavi-Hezavehi et al. [MHAW16], which identifies and categorizes a wealth of works on uncertainty in self-adaptive systems.

The study identifies five dimensions of uncertainty:

1. *Location*: The place in which uncertainty emerges in the self-adaptive system. Identified locations are:
   (a) *Environment*: Context (including interactions with human actors) in which the system is running (e.g., the uncertainties induced by the behavior of a human-in-the-loop, which is not deterministic).
   (b) *Model*: Models that the self-adaptive system employs (typically for decision-making). One example might be the abstraction of some aspect of the real system that is not represented in its model, which induces epistemic uncertainty.
   (c) *Adaptation Functions*: Functionalities that self-adaptive loop performs. An example is the uncertainty caused by faulty sensors of the adaptive system.
   (d) *Goals*: Goals that the self-adaptive loop uses to manage the system. An example is not fully anticipating changing goals in the future.
   (e) *Managed System*: Subsystem being managed by the managing subsystem in the self-adaptive system. An example is uncertainty caused by the complexity of the managed subsystem.
   (f) *Resources*: Components needed by the self-adaptive system to operate. An example is uncertainty from changes in resource availability.
2. *Source*: Subcategories of the location of uncertainty. Table 2.1 describes the different sources of uncertainty identified in [MHAW16].

Class of source of uncertainty	**Options (for sources of uncertainty)**	**Description**	**Example**
Model uncertainty	Abstraction	Uncertainty caused by omitting certain details and information from models for the sake of simplicity.	Simplifying assumptions (Esfahani and Malek [EM13]).
	Incompleteness	Uncertainty caused by parts (of models, mechanisms, etc.) that are knowingly missing because of a lack of (current) knowledge.	Model structural uncertainty(Perez-Palacin et al. [**?**]).
	Model drift	Uncertainty caused by a discrepancy between the state of models and the represented phenomena.	Violation of requirements in models (Ghezzi and Sharifloo [**?**]).
	Different sources of information	Uncertainty caused by differences between the representations of information provided by different sources of information. Uncertainty may be due to different representations of the same information, or result of having different sources of information, or both.	Granularity of models (Cheung et al. [**?**]).
	Complex models	Uncertainty caused by complexity of runtime models representing managed sub systems.	Complex architectural models (Vogel and Giese [**?**]).
Adaptation functions uncertainty	Variability space of adaptation	Uncertainty caused by the size of the variability space that the adaption functions need to handle. This type of uncertainty arises from striving to capture the whole complex relationship of the system with its changing environment in a few architectural configurations which is inherently difficult and generates the risk of overlooking important environmental states [5].	Being unable to foresee all possible environment states as well as all the system configurations in the future (Chauvel et al. [**?**]).
	Sensing	Uncertainty caused by sensors which are inherently imperfect.	Noise in sensing (Esfahani and Malek [EM13]).
	Effecting	Uncertainty caused by sensors which are inherently imperfect.	Future parameter values (Esfahani and Malek [EM13].
	Automatic Learning	Uncertainty caused by machine learning techniques of which the effects may not be completely predictable.	Modeling techniques (Cheung et al., 2007 [**?**])
	Decentralization	Uncertainty due to decision making by different entities of which the effects may not be completely predictable.	Decentralized control in a traffic jams monitoring system (Weyns et al. [**?**]).
	Changes in adaptation mechanisms	Uncertainty due to required dynamicity of adaptation infrastructure to maintain its relevance with respect to the changing adaptation goals (Villegas, Tamura, Müller, Duchien, and Casallas, 2013).	Additional monitoring infrastructure (Villegas et al. [**?**])
	Fault localization and identification	Uncertainty caused by inaccurate localization and identification of faults in the managed system.	Identifying and ranking faulty component (Casanova et al. [**?**])
Goals uncertainty	Goal dependencies	Dependencies between goals, in particular quality goals, may not be captured in a deterministic manner, which causes uncertainty.	Conflict resolution between competing quality attributes (Zoghi et al. [**?**])
	Future goal changes	Uncertainty due to potential changes of goals that could not be completely anticipated.	Rapid evolution (David Garlan [Gar10]).
	Future new goals	Uncertainty due to the potential introduction of new goals that could not be completely anticipated.	Rapid evolution (David Garlan [Gar10])
	Goal specification	Uncertainty due to lack of deterministic specifications of quality goals.	Quality goals priorities changes (Esfahani, et al. [EKM11])

	Outdated goals	Uncertainty caused by overlooking outdated goals.	Addressing goals which are irrelevant to the system (Tamura et al. [**?**])
Environment uncertainty	Execution context	Uncertainty caused by the inherent unpredictability of execution contexts.	Mobility (David Garlan [Gar10])
	Human in the loop	Uncertainty caused by the inherent unpredictability of human behavior.	Objectives (Esfahani and Malek [EM13])
	Multiple ownership	Uncertainty caused by lack of proper information sharing, conflicting goals, and decision making policies by multiple entities that own parts of the system.	Uncertain execution time and failure rate of a component operated by a third-party organization (C Ghezzi et al. [**?**])
Resources uncertainty	New resources	Uncertainty caused by availability of new resources in the system.	Availability of new services in the system (Edwards et al. [**?**]).
	Changing resources	Uncertainty caused by dynamicity of resources in the system.	Resources mobility (Hallsteinsen et al. [**?**])
Managed system uncertainty	System complexity and changes	Uncertainty caused by complexity and dynamicity of nature of the managed system.	Complex systems and complex architectural models (Vogel and Giese [**?**])

Table 2.1: Sources of uncertainty (from [MHAW16]).

3. *Nature*: Specifies whether the uncertainty is epistemic or variable. Epistemic uncertainty is caused by lack of knowledge. An example would be uncertainty introduced while calculating the speed of a falling object without the knowledge of air friction. Here, if air friction is accounted for, the epistemic uncertainty would be handled. Variable uncertainty is due to randomness. An example would be the randomness of human actors in the system.

4. *Level/Spectrum*: On what scale the uncertainty is specified, statistical or scenario based. Statistical uncertainties can be modeled through statistics and probability. Scenarios describe how the system may develop in the future. They do not predict what will happen in the future, but provide possible outcomes.

5. *Emerging Time*: Uncertainties can emerge during design time or runtime. Design time uncertainties arise when the system is being developed, whereas runtime uncertainties arise after the system is deployed.

It is worth noting that the comprehensive classification above does not cover some uncertainties that we identify in this report (such as expiration in Section 3.2).

2.2 Extended Taxonomy of Sources of Uncertainty in Self-Adaptive Systems

In addition to the comprehensive classification of sources of uncertainties presented in table 2.1 (which incorporates references to Esfahani and Malek's categorization described in [EM13]) Ramirez et al. present in [**?**] a different taxonomy for uncertainties in adaptive systems. The latter classification distinguishes three main categories that concern requirements, design, and run-time uncertainties. In this section, we extend the table for sources of uncertainty presented in the previous section, with additional entries obtained from [**?**].

Some of the categories in [**?**] overlap with one or multiple categories in [MHAW16]. Consider for instance categories **A1** and **A2** in the table that concern abstraction and incompleteness. These two categories overlap with category 26 in [**?**] (incomplete information). Even if the latter category is classified in [**?**] as a "run-time" type of uncertainty, part of it can be attributed to the use of incomplete or oversimplified models at run-time in [MHAW16].

The same happens with categories **A3** and **A4** on model drift and different sources of information, which can be mapped to potential inconsistencies (category 25, [**?**]) presented when two or more values of the same properties disagree with each other.

Concerning uncertainties associated with sensing (category **B2**), the classification in [**?**] is much more detailed, distinguishing different cases for sensor failure, noise, imprecision, and inaccuracies (categories 18-21), which are also discussed later on in this report.

With respect to goals, interactions can be given ([**?**], 5) when there are dependencies among them (**C1**). Moreover, future goals and changes on them (C2, C3) may lead to situations in which the goals or requirements defined do not match the actual needs and constraints of the system ([**?**], 8). Goal and requirement specifications can also be ambiguous, potentially leading to problems derived from alternative interpretations (**C4**, [**?**], 2).

Finally, the run-time category of unpredictable environment (22, [**?**]) is general enough to be mapped to different categories of environment uncertainty in [MHAW16], like **D1** which concerns the general unpredictability of the execution context, and **D2**, which narrows down its scope to the context of human behavior predictability, discussed by Esfahani and Malek in [EM13], as well as by Cámara et al. in [CMG15b, **?**].

Class of source of uncertainty	**Options (for sources of uncertainty)**	**Description**	**Example**
A. Model uncertainty	**A1.** Abstraction	Uncertainty caused by omitting certain details and information from models for the sake of simplicity.	Simplifying assumptions (Esfahani and Malek [EM13]), Latency-awareness in adaptation [CMG14], incomplete information [**?**] (26)
	A2. Incompleteness	Uncertainty caused by parts (of models, mechanisms, etc.) that are knowingly missing because of a lack of (current) knowledge.	Model structural uncertainty(Perez-Palacin et al. [**?**]), incomplete information [**?**] (26)
	A3. Model drift	Uncertainty caused by a discrepancy between the state of models and the represented phenomena.	Violation of requirements in models (Ghezzi and Sharifloo [**?**]), inconsistency [**?**] (25).
	A4. Different sources of information	Uncertainty caused by differences between the representations of information provided by different sources of information. Uncertainty may be due to different representations of the same information, or result of having different sources of information, or both.	Granularity of models (Cheung et al. [**?**]), inconsistency [**?**] (25)
	A5. Complex models	Uncertainty caused by complexity of runtime models representing managed sub systems.	Complex architectural models (Vogel and Giese [**?**]).
B. Adaptation functions uncertainty	**B1.** Variability space of adaptation	Uncertainty caused by the size of the variability space that the adaption functions need to handle. This type of uncertainty arises from striving to capture the whole complex relationship of the system with its changing environment in a few architectural configurations which is inherently difficult and generates the risk of overlooking important environmental states [5].	Being unable to foresee all possible environment states as well as all the system configurations in the future (Chauvel et al. [**?**]), latent behavior [**?**] (16)
	B2. Sensing	Uncertainty caused by sensors which are inherently imperfect.	Noise in sensing (Esfahani and Malek [EM13]), sensor failure, noise, imprecision, and inaccuracy [**?**] (18-21)
	B3. Effecting	Uncertainty caused by sensors which are inherently imperfect.	Future parameter values (Esfahani and Malek [EM13].

	B4. Automatic Learning	Uncertainty caused by machine learning techniques of which the effects may not be completely predictable.	Modeling techniques (Cheung et al., 2007 [**?**])
	B5. Decentralization	Uncertainty due to decision making by different entities of which the effects may not be completely predictable.	Decentralized control in a traffic jams monitoring system (Weyns et al. [**?**]).
	B6. Changes in adaptation mechanisms	Uncertainty due to required dynamicity of adaptation infrastructure to maintain its relevance with respect to the changing adaptation goals (Villegas, Tamura, Müller, Duchien, and Casallas, 2013).	Additional monitoring infrastructure (Villegas et al. [**?**])
	B7. Fault localization and identification	Uncertainty caused by inaccurate localization and identification of faults in the managed system.	Identifying and ranking faulty component (Casanova et al. [**?**])
C. Goals uncertainty	**C1.** Goal dependencies	Dependencies between goals, in particular quality goals, may not be captured in a deterministic manner, which causes uncertainty.	Conflict resolution between competing quality attributes (Zoghi et al. [**?**]), requirements interaction [**?**] (5)
	C2. Future goal changes	Uncertainty due to potential changes of goals that could not be completely anticipated.	Rapid evolution (David Garlan [Gar10]), changing requirements [**?**] (8).
	C3. Future new goals	Uncertainty due to the potential introduction of new goals that could not be completely anticipated.	Rapid evolution (David Garlan [Gar10]), changing requirements [**?**] (8)
	C4. Goal specification	Uncertainty due to lack of deterministic specifications of quality goals.	Quality goals priorities changes (Esfahani, et al. [EKM11]), ambiguous requirements [**?**] (2)
	C5. Outdated goals	Uncertainty caused by overlooking outdated goals.	Addressing goals which are irrelevant to the system (Tamura et al. [**?**])
D. Environment uncertainty	**D1.** Execution context	Uncertainty caused by the inherent unpredictability of execution contexts.	Mobility (David Garlan [Gar10]), unpredictable environment [**?**] (22)
	D2. Human in the loop	Uncertainty caused by the inherent unpredictability of human behavior.	Objectives (Esfahani and Malek [EM13]), capabilities, willingness [CMG15b, **?**]
	D3. Multiple ownership	Uncertainty caused by lack of proper information sharing, conflicting goals, and decision making policies by multiple entities that own parts of the system.	Uncertain execution time and failure rate of a component operated by a third-party organization (C Ghezzi et al. [**?**])
E. Resources uncertainty	**E1.** New resources	Uncertainty caused by availability of new resources in the system.	Availability of new services in the system (Edwards et al. [**?**]).
	E2. Changing resources	Uncertainty caused by dynamicity of resources in the system.	Resources mobility (Hallsteinsen et al. [**?**])
F. Managed system uncertainty	**F1.** System complexity and changes	Uncertainty caused by complexity and dynamicity of nature of the managed system.	Complex systems and complex architectural models (Vogel and Giese [**?**])

Table 2.2: Extended sources of uncertainty (from [MHAW16, **?**]).

2.3 Theories and Techniques for Uncertainty Management in Self-Adaptive Systems

2.3.1 Fuzzy Sets and Possibility Theory

Fuzzy set theory extends classical set theory with the concept of *degree of membership* [Zad65], making its use appropriate for domains in which information is imprecise or incomplete. Rather than assessing the membership of an element to a set in binary terms (an element belongs to a set or not), fuzzy set theory describes membership as a function in the real interval [0,1], where values closer to 1 indicate higher likelihood of the element belonging to the set. Possibility theory [Zad99] is based on fuzzy sets, and in its basic interpretation, it assumes that given a finite set (e.g., describing possible future states of the world), a *possibility distribution* is as a mapping between its power set, and the real interval $[0, 1]$ (i.e., any subset of the sample space has a possibility assigned by the mapping).

In the context of self-adaptive systems, possibility theory has been mainly used in approaches that deal with the uncertainty of the objectives [BPS10, EKM11, WSB+09]. RELAX [WSB+09] is a formal specification language for requirements that employs possibility theory to account for the uncertainty in the objectives of the self-adaptive system. The language introduces a set of operators that allows the "relaxation" of requirements at run time, depending on the state of the environment. FLAGS [BPS10] also employs possibility theory to mitigate the uncertainty derived by the environment and changes in requirements by embedding adaptability at requirements elicitation. In particular, the framework introduces the concept of *adaptive goals* and *counter measures* that have to be executed if goals are not satisfied as a result of predicted uncertainty. POISED [EKM11] is a quantitative approach that employs possibility theory to assess the positive and negative consequences of uncertainty. The approach incorporates a framework that can be tailored to specify the relative importance of the different aspects of uncertainty, enabling developers to specify a range of decision-making approaches, e.g., favoring adaptations that provide better guarantees in worst-case scenarios against others that involve higher risk but better maximization of the expected outcome.

2.3.2 Probability Theory

Probability theory [Kol56] is the branch of mathematics concerned with the study of random phenomena. Probability is the measure of the likeliness that an event will occur, and is quantified as a number in the real interval $[0, 1]$ (where 0 indicates impossibility and 1 certainty). Within probability theory, frequentist interpretations of random phenomena employ information relative to the frequencies of past *actual* outcomes to derive probabilities that represent the *likelihood* of possible outcomes for future events.

This interpretation of probability is widely employed to deal with different sources of uncertainty in self-adaptive systems (e.g., context, simplifying assumptions, model drift) [EEM10, GCH+04]. FUSION [EEM10] is an approach to constructing self-adaptive systems that uses machine learning to tune the adaptive behavior of a system in the presence of unanticipated changes in the environment. The learning focuses on the relation between adaptations, effects and system qualities, helping to mitigate uncertainty by considering explicitly interactions between adaptations. Rainbow [GCH+04] is an approach to engineering self-adaptive systems that includes constructs to deal with the mitigation of uncertainty in different activities of the MAPE loop [KC03]. In particular, the framework employs running averages to mitigate uncertainty due to noise in monitoring, as well as explicit annotation of adaptation strategies with probabilities (obtained from past observations of the running system) to account for uncertainty when selecting strategies during the planning stage.

Moreover, other approaches employ probabilistic verification and estimates of the future environment and system behavior for optimizing the system's operation. These proposals target the mitigation of uncertainty due to parameters over time [C+11, CK09, E+09]. Calinescu and Kwiatkowska [CK09] introduce an autonomic architecture that uses Markov-chain quantitative analysis to dynamically adjust the parameters of an IT system according to its environment and goals. Epifani *et al.* introduce KAMI [E+09], a methodology and framework to keep models alive by feeding them with run-time data that updates their internal parameters. The framework focuses on reliability and performance, and uses discrete-time Markov chains and

[1] POISED employs probability theory to mitigate uncertainty due to noise.

		Source of Uncertainty								
Theory	**Approach**	**Simplifying Assumptions**	**Model Drift**	**Noise**	**Parameters over time**	**Human in the loop**	**Objectives**	**Decentralization**	**Context**	**Cyber-physical systems**
Fuzzy Sets / Possibility Theory	RELAX [WSB+09]						✓			
	FLAGS [BPS10]						✓			
	POISED [EKM11]	✓		✓[1]			✓			
Probability Theory	Rainbow [GCH+04]	✓		✓						
	FUSION [EEM10]	✓	✓						✓	
	Calinescu and Kwiatkowska [CK09]				✓					
	KAMI [EGMT09]				✓					
	QoSMOS [C+11]				✓					
	MAUS [ES11]					✓				
Probability + Game Theory	Li et al. [LSSS14]					✓				
	Emami-Taba et al. [ETAT15]				✓					
	Cámara et al. [CMG14, CMG15a]	✓			✓	✓				

Table 2.3: Theories and approaches to mitigate uncertainty in self-adaptive systems

queuing networks as models to reason about the evolution of non-functional properties over time. Moreover, the QoSMOS framework [C+11] extends and combines [CK09] and [E+09] to support the development of adaptive service-based systems. In QoSMOS, QoS requirements are translated into probabilistic temporal logic formulae used for identifying and enforcing optimal system configurations.

Eskins and Sanders introduce in [ES11] a Multiple-Asymmetric-Utility System Model (MAUS) and the opportunity-willingness-capability (OWC) ontology for classifying cyber-human systems elements with respect to system tasks. This approach provides a structured and quantitative means of analyzing cyber security problems whose outcomes are influenced by human-system interactions, dealing with the uncertainty derived from the probabilistic nature of human behavior.

2.3.3 Probability and Game Theory

Game theory is the study of mathematical models of conflict and cooperation between intelligent rational decision-makers [Mye91]. The theory studies situations where there are multiple decision makers or *players* who have a variety of of alternatives or *strategies* to employ in order to achieve a particular outcome (e.g., in terms of loss or payoff). Game theory has been applied in a wide variety of fields, such as, Economics, Biology, and Computer Science to study systems that exhibit competitive behavior (e.g., zero-sum games in which the payoff of a player is balanced by the loss of the other players), as well as a range of scenarios that might include cooperation (e.g., when players in a coalition coordinate to choose joint strategies by consensus).

Li et al [LSSS14] present a formalism for human-in-the-loop control systems aimed at synthesizing semi-autonomous controllers from high-level temporal specifications (LTL) that expect occasional human intervention for correct operation. The approach adopts a game-theoretic approach in which controller synthesis is performed based on a (non-stochastic) zero-sum game played between the system and the environment. Although this proposal deals to some extent with uncertainty due to parameters over time and human involvement, the behavior of all players in the game is specified in a fully nondeterministic fashion, and once

nondeterminism is resolved by a strategy, the outcome of actions is deterministic.

Emami-Taba et al. [ETAT15] present a game-theoretic approach that models the interplay of a self-protecting system and an attacker as a two-player zero-sum Markov game. In this case, the approach does not perform any exhaustive state-space exploration and is evaluated via simulation, emphasizing the learning aspects of the interaction between system and attacker.

Stochastic Multiplayer Games (SMGs) [CFK+13] are models that fit naturally systems that exhibit both probabilistic and competitive behavior among different players who can either collaborate with each other, or compete to achieve their own goals. In prior work, we presented in [CMG14] an analysis technique based on model checking of SMGs to quantify the effects of simplifying assumptions in proactive self-adaptation. Specifically, the paper shows how the technique enables the comparison of alternatives that consider tactic latency information for proactive adaptation with those that are latency-agnostic, making the simplifying assumption that tactic executions are not subject to latency (i.e., that the duration of the time interval between the instants in which a tactic is triggered and its effects occur is zero). In [CMG15a] we adapted this analysis technique to apply it in the context of human-in-the-loop adaptation, extending SMG models with elements that encode an extended version of Stitch adaptation models [CG12] with OWC constructs [ES11].

We present in the following section SMGs in more detail, since they are the formal foundation that we employ in the approach described in this chapter.

2.3.4 Probabilistic Model Checking of Stochastic Multiplayer Games

Automatic verification techniques for probabilistic systems have been successfully applied in a variety of application domains including security [DKSS14, SCG+14] and communication protocols [HZHC15]. In particular, techniques such as probabilistic model checking provide a means to model and analyze systems that exhibit stochastic behavior, effectively enabling reasoning quantitatively about probability and reward-based properties (e.g., about the system's use of resources, time, etc.).

Competitive behavior may also appear in systems when some component cannot be controlled, and could behave according to different or even conflicting goals with respect to other components in the system. Self-adaptive systems are a good example of systems in which the behavior of some components that are typically considered as part of the environment (non-controllable software, network, human actors) cannot be controlled by the system. In such situations, a natural fit is modeling a system as a game between different players, adopting a game-theoretic perspective.

Our approach to analyzing self-adaptation builds upon a recent technique for modeling and analyzing stochastic multi-player games (SMGs) extended with rewards [CFK+13]. In this approach, systems are modeled as turn-based SMGs, meaning that in each state of the model, only one player can choose between several actions, the outcome of which can be probabilistic. This approach is particularly promising, since SMGs are amenable to analysis using model checking tools, and are expressive enough to capture: (i) the uncertainty and variability intrinsic to the execution environment of the system in the form of probabilistic and nondeterministic choices, and (ii) the competitive behavior between the system and the environment, which can be naturally modeled as players in a game whose behavior is independent (reflecting the fact that changes in the environment cannot be controlled by the system).

Definition 1 (SMG)**.** *A turn-based stochastic multi-player game augmented with rewards (SMG) is a tuple* $\mathcal{G} = \langle \Pi, S, A, (S_i)_{i\in\Pi}, \Delta, AP, \chi, r\rangle$*, where* Π *is a finite set of players;* $S \neq \emptyset$ *is a finite set of states;* $A \neq \emptyset$ *is a finite set of actions;* $(S_i)_{i\in\Pi}$ *is a partition of* S*;* $\Delta : S \times A \to \mathcal{D}(S)$ *is a (partial) transition function;* AP *is a finite set of atomic propositions;* $\chi : S \to 2^{AP}$ *is a labeling function; and* $r : S \to \mathbb{Q}_{\geq 0}$ *is a reward structure mapping each state to a non-negative rational reward.* $\mathcal{D}(X)$ *denotes the set of discrete probability distributions over finite set* X*.*

In each state $s \in S$ of the SMG, the set of available actions is denoted by $A(s) = \{a \in A \mid \Delta(s, a) \neq \bot\}$. We assume that $A(s) \neq \emptyset$ for all states s in the model. Moreover, the choice of which action to take in every state s is under the control of a single player $i \in \Pi$, for which $s \in S_i$. Once action $a \in A(s)$ is selected by a player, the successor state is chosen according to probability distribution $\Delta(s, a)$.

Definition 2 (Path)**.** *A path of SMG* $\mathcal{G}$ *is an (in)finite sequence* $\lambda = s_0 a_0 s_1 a_1 \ldots$ s.t. $\forall j \in \mathbb{N} \bullet a_j \in A(s_j) \wedge \Delta(s_j, a_j)(s_{j+1}) > 0$*. The set of all finite paths in* $\mathcal{G}$ *is denoted as* $\Omega^+_{\mathcal{G}}$*.*

Players in the game can follow strategies for choosing actions in the game, cooperating with each other in coalition to achieve a common goal, or competing to achieve their own (potentially conflicting) goals.

Definition 3 (Strategy)**.** *A strategy for player $i \in \Pi$ in $\mathcal{G}$ is a function $\sigma_i : (SA)^* S_i \to \mathcal{D}(A)$ which, for each path $\lambda \cdot s \in \Omega^+_{\mathcal{G}}$ where $s \in S_i$, selects a probability distribution $\sigma_i(\lambda \cdot s)$ over $A(s)$.*

In the context of our approach, we always refer to player strategies σ_i that are *memoryless* (i.e., $\sigma_i(\lambda \cdot s) = \sigma_i(\lambda' \cdot s)$ for all paths $\lambda \cdot s$, $\lambda' \cdot s \in \Omega^+_{\mathcal{G}}$), and *deterministic* (i.e., $\sigma_i(\lambda \cdot s)$ is a Dirac distribution for all $\lambda \cdot s \in \Omega^+_{\mathcal{G}}$). Memoryless, deterministic strategies resolve the choices in each state $s \in S_i$ for player $i \in \Pi$, selecting actions based solely on information about the current state in the game. These strategies are guaranteed to achieve optimal expected rewards for the kind of cumulative reward structures that we use in our models.[2]

Reasoning about strategies is a fundamental aspect of model checking SMGs, which enables checking for the existence of a strategy that is able to optimize an objective expressed as a property in a logic called rPATL. Concretely, rPATL can be used for expressing quantitative properties of SMGs, and extends the logic PATL [CL07] (a probabilistic version of ATL [A$^+$02], a logic extensively used in multi-player games and multi-agent systems to reason about the ability of a set of players to collectively achieve a particular goal). Properties written in rPATL can state that a coalition of players has a strategy which can ensure that the probability of an event's occurrence or an expected reward measure meets some threshold.

rPATL is a CTL-style branching-time temporal logic that incorporates the coalition operator $\langle\langle C \rangle\rangle$ of ATL, combining it with the probabilistic operator $\mathsf{P}_{\bowtie \mathsf{q}}$ and path formulae from PCTL [BdA95]. Moreover, rPATL includes a generalization of the reward operator $\mathsf{R}^{\mathsf{r}}_{\bowtie \mathsf{x}}$ from [F$^+$11] to reason about goals related to rewards. An extended version of the rPATL reward operator $\langle\langle C \rangle\rangle \mathsf{R}^{\mathsf{r}}_{\mathsf{max}=?}[\mathsf{F}^{\star}\ \phi]$ [3] enables the quantification of the maximum accrued reward r along paths that lead to states satisfying state formula ϕ that can be guaranteed by players in coalition C, independently of the strategies followed by the rest of players. An example of typical usage combining the coalition and reward maximization operators is $\langle\langle \mathsf{sys} \rangle\rangle \mathsf{R}^{\mathsf{utility}}_{\mathsf{max}=?}[\mathsf{F}^{\mathsf{c}}\ \mathsf{end}]$, meaning "value of the maximum utility reward accumulated along paths leading to an end state that a player sys can guarantee, regardless of the strategies of other players."

2.4 Uncertainty Management in other Areas

Uncertainty management is not exclusive to research in self-adaptive systems. Other related areas, such as artificial intelligence, or research in cyber-physical systems have dealt with this topic in different works. In this section, we overview a non-exhaustive compilation of related work in these areas.

2.4.1 Cyber-Physical Systems

Uncertainty is inherent to Cyber-Physical Systems (CPS). Hence, handling uncertainty in a graceful manner during the operation of a CPS is critical. Designing, developing, and testing modern and sophisticated CPS requires dealing comprehensively with the different types of uncertainty that might arise during system operation. This requires identifying, defining, and classifying uncertainties at various levels in CPS.

To achieve that goal, the authors of [**?**] propose a conceptual model for uncertainty specifically tailored for CPS. This conceptual model is derived mostly by reviewing existing work on uncertainty in other fields, including philosophy, physics, statistics, and healthcare. The conceptual model is mapped to the three logical levels of CPS: Application, Infrastructure, and Integration. It is captured using UML class diagrams, including relevant OCL constraints. To validate the conceptual model, the authors identified, classified, and specified uncertainties in two different industrial case studies.

From the perspective of security, uncertainties derived from sensing and actuating (Table 2.2, categories **B2** and **B3**) emerge as some of the main concerns in CPS due to the use that potential attackers might make of compromised sensors and effectors to steer a system into unsafe states that might ultimately have an impact not only on the software, but also on the physical context of the system.

[2] See Appendix A.2 in [CFK$^+$13] for details.

[3] The variants of $\mathsf{F}^{\star}\phi$ used for reward measurement in which the parameter $\star \in \{0, \infty, c\}$ indicate that, when ϕ is not reached, the reward is zero, infinite or equal to the cumulated reward along the whole path, respectively.

The work in [?] is concerned with the estimation and control of linear systems when some of the sensors or actuators are corrupted by an attacker. The authors of [?] tackle a similar problem, with a stronger focus on sensing and state estimation in continuous-time linear systems, for which an attacker may have control over some of the sensors and inject (potentially unbounded) additive noise into some of the measured outputs. To characterize the resilience of a system against such sensor attacks, the authors introduce a notion of "observability under attacks" that addresses the question of whether or not it is possible to uniquely reconstruct the state of the system by observing its inputs and outputs over a period of time, with the understanding that some of the available system's outputs may have been corrupted by the attacker.

The authors of [?] study CPS subject to dynamic sensor attacks, relating them to the system?s strong observability. This work identifies necessary and sufficient conditions for an attacker to create a dynamically undetectable sensor attack and relate these conditions to properties of the system dynamics. Moreover, the authors also provide an index that gives the minimum number of sensors that must be attacked in order for an attack to be undetectable.

On a related line of work, the authors of [?] study the effects of uncertainty in models induced by unverified assumptions about the trustworthiness of physical devices, such as sensors. When these assumptions are violated, subtle inter-domain vulnerabilities are introduced into the system model. This study explores the use formal specification of analysis contracts to expose security assumptions and guarantees of analyses from reliability, control, and sensor security domains.

2.4.2 Artificial Intelligence

Researchers in artificial intelligence have been working on approaches for reasoning under uncertainty for a very long time, compared to other areas like self-adaptive systems or CPS. There is a wealth of paradigms, algorithms, and techniques that is too vast to summarize in this chapter. However, notable approaches to reasoning under uncertainty include the use of probabilistic planners [?] that deal with aleatoric uncertainty and even epistemic uncertainty via reasoning about partial observability [?].

For a more comprehensive compilation of techniques for uncertainty management in artificial intelligence, we refer the reader to a recent survey that focuses on this topic [?].

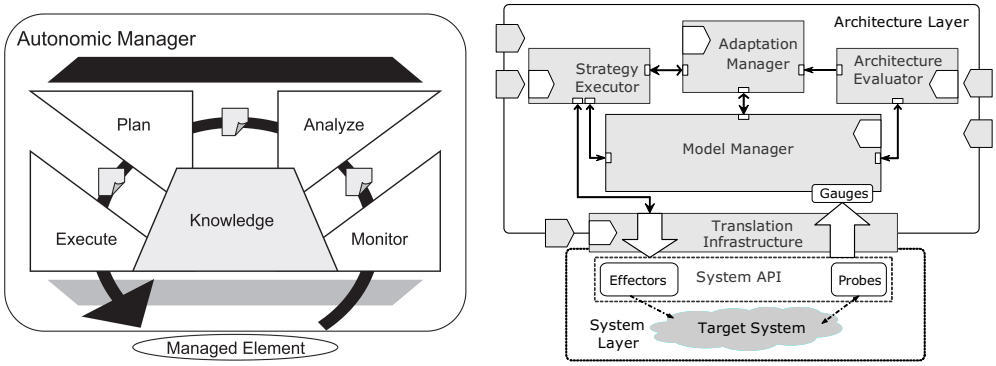
Chapter 3

Representing Uncertainty

Self-adaptation is a promising way of engineering systems able to achieve run-time dependability and resilience by adapting their structure and behavior in the face of unexpected changes. Although there are different paradigms to build self-adaptation into software-intensive systems, one of the most popular models is IBM's MAPE-K (Figure 3.1 (a)). The MAPE-K model integrates in a closed-control loop activities to monitor, analyze, plan, and execute adaptations by effecting changes in a managed software (sub)system. Moreover, a central knowledge base that typically includes models about the managed system [CG12], its environment, and adaptations, informs the different MAPE activities.

Rainbow is a self-adaptation framework with an emphasis on system architecture [CG12] that instantiates the MAPE-K model. Figure 3.1(b) describes how in Rainbow, *probes* monitor the system and *gauges* aggregate the monitored information and update the architecture-centric model of the system. The *models manager* instantiates the K of MAPE-K in Rainbow, where one of the core models describing the managed system is its architecture model, specified in the Acme architecture description language (ADL) [**?**]. The *architecture evaluator* then detects if the system is in a state where adaptation (e.g., repair) is needed, and the *adaptation manager* selects an appropriate strategy to adapt the managed system. The adaptation manager defines reactive adaptation strategies using the Stitch language [CG12]. Finally, the *strategy executor* carries out the changes described in the strategy via *effectors* embedded in the managed system.

The models employed for decision-making in self-adaptive systems are subject to different types of uncertainty (e.g., reading from sensors might be inaccurate, some aspect of the domain may be abstracted away in models). However, despite the fact that these uncertainties can have a noticeable impact on run-time system behavior, many approaches to engineering self-adaptation do not model the uncertainties that affect the system explicitly or as a first-class entity [Gar10]. In the case of the Rainbow framework, some aspects of uncertainty are addressed. For example, uncertainty from human-in-the-loop interactions are addressed via explicit modeling and reasoning in [CMG15b], whereas other uncertainties, like those derived from noisy



(a) Mape-K loop.

(b) Rainbow framework.

Figure 3.1: MAPE-K loop and the Rainbow framework.

sensor readings are implicitly dealt with via smoothing through a running average in gauges (the components that gather information from sensors and feed it to the models manager) [Gar10]. Moreover, decision-making based on probabilistic models of the managed system, its environment, and the effects of adaptations have been explored to address some sources of epistemic uncertainty (e.g., latency in adaptation tactics [CMG14]). Nevertheless, there is no comprehensive work that exhaustively compiles the different sources and constructs to model uncertainty that affect a typical MAPE-K system.

In this chapter, we describe a systematic approach for representing uncertainty in MAPE-K self-adaptive systems, employing Rainbow as a vehicle to illustrate the proposal. Section 3.1 gives an overview of the different sources of uncertainty that may affect different parts of MAPE-K. Next, Section 3.2, goes through different types of uncertainties identified in the core models employed by Rainbow, proposing constructs in the Acme ADL and extensions of the description language for their specification. Section 3.3 identifies some of the uncertainty types described in this chapter on an example self-adaptive system (Znn.com, an enterprise web infrastructure with MAPE-K adaptive capabilities built using Rainbow).

3.1 Uncertainties in MAPE-K

We can identify different types of uncertainty in parts of the MAPE-K loop.

1. *Monitoring.* Identified by Esfahani and Malek [EM13], sensing in MAPE-K can be affected by different uncertainties related to the inherent imperfections of sensors.
    - Sensing latency. Information can take time to be processed by the monitoring infrastructure. By the time it is processed by analysis it may be outdated.
    - Sensing inaccuracy. There are aleatoric uncertainties induced by inaccuracies in sensor readings (i.e., deviations from the ideal reading of the sensor). This is also related to sensor *resolution*, which can also introduce uncertainties because the sensor cannot distinguish differences in readings that are smaller than a given threshold.
    - Sensing reliability. Sensors may fail, providing values that are far from the real state of the world, or no values at all. For instance, if a sensor breaks, it may start providing values entirely different from those that would be reported under normal operation.
2. *Analysis and Planning.* Analysis and planning can be affected by different types of uncertainties, mostly related to models employed to reason about the need to adapt, and how to adapt. However, there are additional uncertainties introduced that concern the reasoning mechanisms themselves, such as those that affect the time required to produce a decision in the adaptation loop.
    - Analysis/Planning latency. Time it takes to complete analysis/planning. Particularly important when sophisticated planning for solution synthesis takes place (e.g., MDP solving).
    - Model uncertainty. There are different uncertainties induced by models that directly affect the analysis and planning stages of MAPE-K. Those are summarized in Table 2.1, and concern model abstraction, incompleteness, complexity, drift, as well as different sources of information.
3. *Execution.* There are different uncertainties that affect execution and are in general related to the lack of determinism in effecting changes in the managed subsystem. These are aligned with the *future parameter values* source identified by Esfanani and Malek [EM13].
    - Execution Latency (or execution time). The time between the instant in which adaptation execution is triggered, and the instant in which it is actually carried out (i.e., execution *latency*)), will not be deterministic in the general case.
    - Effector reliability. The effector will not always successfully execute its required action. Instead, there is always a probability (even if it is a small one) that the effector will fail to correctly execute its action (e.g., a server that is being booted in the managed subsystem may crash).

In the next section, we are going to describe how some of the types of uncertainties described above can be formally captured in architecture-centric models, employing the Acme ADL to illustrate our proposal.

3.2 Representing Uncertainties

The Rainbow framework uses the Acme architecture description language to describe the model of the system that it manages. However, although some types of uncertainties can be captured in the current version of Acme, there is not a comprehensive catalog of constructs patterns to support the systematic specification of different types of uncertainty. In this section, we propose a catalog of such constructs and patterns to extend Acme's capabilities in terms of capturing different types of uncertainty.

3.2.1 Functions

We first introduce the different types of function that we use to model uncertainty (Table 3.1). We chose this reduced set of functions because they are able to capture a wide array of probabilistic aspects of a system in a simple way The functions that we specify here are used to model common probability distributions, making them versatile and reusable across different types of specification of uncertainty. Although this set can be extended with additional functions or distributions, the three basic functions described in Table 3.1 form a core expressive enough to capture the types of uncertainty considered in this report.

Linear function	Normal distribution	Exponential distribution

Property Type LinearFunctionT = **Record** [
  slope : **Float**;
  intercept : **Float**;
];
	Property Type NormalDistributionT = **Record** [
	  mean: **Float**;
	  stddev: **Float**;
	];
		Property Type ExponentialDistributionT = **Record** [
		  rate : **Float**;
		];

Table 3.1: Core set of functions employed to capture different types of uncertainty in Acme.

Based on that core set of functions, we can represent 13 different types of uncertainty that we have identified in Rainbow Table 3.2. In the table, we map each type of uncertainty to the different dimensions described in Section 2.1. However, note that since Acme is used to model just the managed subsystem, not all dimensions of uncertainty are represented in this table. For example, goal uncertainties have been left out of scope, since in Rainbow, Stitch models are responsible for managing the goals of the system. Concretely, we identify uncertainties in the broad categories of sensors, effectors, and human-in-the-loop interactions.

Uncertainty	Location	Source	Nature	Level	Emerging Time	Example
Range	Adaptation Functions	Sensing	Epistemic	Statistical	Run time	Thermometer
Accuracy						
Resolution						
Response Time			Variable			CPU Usage
Error Distribution						
Expiration						
Aggregate			Epistemic			Response Time (based on CPU load and queue length)
Success Rate		Effecting	Variable			Adding a new server
Expected Execution Time						
Opportunity	Environment	Human in the Loop	Epistemic			Operators of a power plant
Willingness						
Capability			Variable			
Human Latency						

Table 3.2: Classification of uncertainties modeled in Acme.

3.2.2 Sensors

We first begin with specifying standard sensor specifications found in physical sensors. We use a thermometer as a running example for specifications. As Table 3.2 indicates, the following uncertainties are epistemic in nature, can be modeled statistically, and emerge during run time.

- *Range.* The range of a sensor represents the lowest and highest sensor readings the sensor can register. This information can be used to make sure that a given sensor reading is valid (i.e., within range). The range of a thermometer corresponds to the magnitudes between the lowest and highest temperature readings that the thermometer can register.

```
Property Type SensorRangeT = Record [
    min: Float;
    max: Float;
];
```

- *Accuracy.* Accuracy represents the deviations in sensor readings from real magnitude values. For example, if the accuracy is 1 degree celsius for a thermometer, this means that if the thermometer reads 20 degrees, the actual temperature could be between 19 and 21 degrees celsius.

```
Property Type SensorAccuracyT = Float;
```

- *Resolution.* Resolution represents the minimum sensor reading difference that the sensor can distinguish (i.e., its finest granularity). If the resolution of the thermometer is 1 degree, then the thermometer may not sense a difference between 20 degrees and 20.5 degrees. Note that accuracy and resolution are different, since a thermometer may not be very accurate but be able to sense small differences in temperature.

```
Property Type SensorResolutionT = Float;
```

We now model response time, error distribution, and expiration. Unlike the previous specifications, the nature of these types of uncertainty is variable. We will use a CPU usage sensor as a running example.

- *Response Time.* The response time represents how long it takes the given sensor to collect information and produce a sensor reading (let us assume that time is measured milliseconds in our example). Note that we model response time using a normal distribution in order to take into account the variable nature of sensors, since there could be times in which the sensor takes more or less time to produce a reading. In the case of CPU usage, the response time could for example be centered at 100 ms with a standard deviation of 10 ms.

```
Property Type SensorResponseTimeT = NormalDistributionT;
```

- *Error Distribution.* Error distribution represents the distribution of error of the sensor readings. This value is related to our accuracy specification, only specified as a normal distribution. Accuracy is a hard value that sensor manufacturers specify, whereas the error distribution is modeled as a normal distribution to take into account the variable error values. In CPU usage, we know that the readings are very noisy, so the error distribution typically has a large standard deviation. Error distributions are suitable to be employed by Kalman filters [**?**] that can be incorporated in gauges of the Rainbow framework. Kalman filters allow sensor values to be smoothened out to reduce the noise in sensor readings.

```
Property Type SensorErrorDistributionT = NormalDistributionT;
```

- *Expiration.* Expiration of a sensor reading represents the length of the timespan during which a reading is valuable, since the time instant in which it was taken. For example, in CPU usage we know that a value from a second ago is unlikely to be valuable. In order to use expiration, we first provide a type to keep the timestamp of a given sensor reading. We then model the value of a sensor reading as a either a scalar value, a linear function, or an exponential distribution. Scalar values indicate that a reading becomes invalid after a given time lapse. Linear functions represent a linear degradation in value over time. Exponential distributions model an alternative progressive value degradation. A potential use for this kind of specification is factoring it into decision-making, providing higher weight in the decision to "fresher" pieces of information.

```
Property Type SensorDataTimestampT = Integer;
Property Type SensorExpirationT =
    Record [
        type: Enum { scalar, linear, exponential };
        scalar : Integer;
        linear : LinearFunctionT;
        exponential: ExponentialDistributionT;
    ];
```

- *Aggregate.* The aggregate specification can be used to mitigate uncertainties for a given sensor reading by aggregating different sensor readings in order to estimate a sensor value. This can be done by using Kalman filters. Specifying the covariance between the different sensor readings, we can generate better estimates of the sensor value. For example, consider that we have separate sensors to measure the response time, CPU usage, and queue length of a server. We know that all three are related, so their registered values will have a given covariance. A Kalman filter can provide an estimate response time based on the cpu usage, queue length and response time readings. Another use case for the aggregate specification could be for sensors that have a high cost to run. For instance, if a sensor A takes a long time to register a reading, we can employ cheaper sensors B and C to estimate a value for sensor A via Kalman filters.

```
Property Type SensorAggregateT =
    Sequence <
        Record [
            sensor: String;
            covariance: Float;
        ]
    >;
```

3.2.3 Effectors

For effectors, we identify two sources of uncertainty (both variable in nature): success rate and expected execution time. We employ server activation as an example to illustrate these uncertainties.

- *Success Rate.* The success rate of an effector would be the probability at which the effector successfully executes its action. For the example of adding a new server, it could be the case that the action succeeds 90% of the time. An extension to this construct entails defining functions for the success rate, as opposed to a static value. This construct can be employed for instance to allocate a new server in Amazon Web Services, where the chances of getting a given server varies depending on the time of the day, geographical area, or system load. A more comprehensive, context-sensitive approach that captures this kind of uncertainty at the level of adaptation tactics is described in [**?**].

```
Property type EffectorSuccessRateT = Float;
```

- *Expected Execution Time.* The expected execution time represents the time it takes for the effector to complete its action. Now, it is important to note that this is different from the expected time it takes for the effectors to have an impact on the system. We represent the value as a normal distribution to model the variability of the effectors at run-time. For adding a server, there is an average time in which we expect the server to start up, and a standard deviation, with an associated standard deviation.

```
Property type EffectorExecutionTimeT = NormalDistributionT;
```

3.2.4 Human in the Loop

For specifying uncertainties in human-in-the-loop adaptive systems, we have reworked the specifications of opportunity, willingness, and capability inspired by Eskins and Sanders work [ES11], and incorporated as an extension of Stitch described in [CMG15b]. A part of the human models described in that work can naturally reside in Acme specifications, and be referenced for decision-making from Stitch specifications. For human-in-the-loop uncertainty specifications, we employ a human operator in a power plant as a running example.

- Opportunity. Captures the applicability conditions of the adaptation tactics that can be executed by human actors upon the target system, as constraints imposed on the human actor (e.g., by the physical context – is there an operator physically located on site?). In the Rainbow framework, opportunity for a human actor to act in the system is represented by two boolean values [CMG15b]. We make sure that the human operator is on site and not busy. For example, an operator in a power plant would be unable to act for the system if she is away or busy with another task. It is important to note that these values themselves have uncertainty, because the system does not know with absolute certainty that the operator is on site. However, these specifications can be refined by substituting boolean values by probabilities.

```
Property Type OnLocationT = Boolean;
Property Type BusyT = Boolean;
```

- Willingness. Captures transient factors that might affect the disposition of the operator to carry out a particular task (e.g., load, stamina, stress). Continuing with our example, if the operator of a power plant is very stressed, she would be less willing to perform a task. Since there is uncertainty induced by the measurement the training level, we represent it as a normal distribution.

```
Property Type CurrentStressLevelT = NormalDistributionT;
```

- Capability. Captures the likelihood of successfully carrying out a particular task, which is determined by fixed attributes of the human actor, such as training level. Just like in the case of willingness, there is an uncertainty inherent to the measurement process. Hence, we represent the training level in our example as a normal distribution.

```
Property Type TrainingLevelT = NormalDistributionT;
```

We group the human-in-the-loop specifications into a single type. We reuse the execution time specification from effectors, so that we can model the time it takes for the human actor to respond to the system. Note that opportunity and willingness are defined per operator, whereas capability and expected execution time is defined per action.

```
Property Type OperatorT =
    Record [
        onLocation:       OnLocationT;
        busy:             BusyT;
        currentStressLevel: CurrentStressLevelT;
        Actions: Sequence <
            Record [
                Action:        String;
                trainingLevel: TrainingLevelT;
                latency:       EffectorExecutionTimeT;
            ];
        >
    ];
```

3.2.5 Structural

Structural uncertainty occurs when the self-adaptive system's internal representation of the managed subsystem is inconsistent with the actual system. An example would be, how do we know if an "Add Serve" action actually added a server as opposed to another client? Another example would be determining if a given connector in the system actually connects the two components that the model indicates.

It is important to distinguish two different types of structural uncertainty.

- *Design time.* When the model of the system is specified, it could be inaccurate, introducing structural uncertainty. The location of the uncertainty is the model, nature is epistemic, level is statistical (since we could measure the difference between the model and the system), and emerging time is design time. Modeling the first type of structural uncertainty in Acme specifications would not be practical in Rainbow, since the Acme specification itself is the model that the managing subsystem uses.

- *Run time.* In this case, the sources of uncertainty are run-time actions, or the sensors employed to monitor the managed subsystem. The classification of the uncertainty is the same as in design-time structural uncertainty, except for the emerging time, which is run time, and nature (variable). In order to mitigate run-time structural uncertainty, additional sensors can be incorporated throughout the system to sense aspects relevant to its structure. For example, we can have sensors on servers to report their status. So when an "Add Server" action is run, we can listen to the new server's status sensor to make sure that the server has indeed been added. We then can reuse the sensor uncertainty specifications described in Section 3.2.2 to model run-time structural uncertainties. As a result, we do not require additional uncertainty specifications to capture this type of structural uncertainty.

3.3 Identifying Uncertainties in Znn.com

To illustrate the use of uncertainty specifications, we will use Znn.com as an example. Znn.com is a simple news service that uses the Rainbow framework to deal with DDoS attacks through different tactics [SCG+14]. As figure 3.2 indicates, Znn.com has several servers, where some are inactive, a load balancer, and a database. Rainbow monitors the load balancer, servers, and the database to update the Acme model of Znn.com to make decisions. In order to adapt to DDoS attacks, Rainbow executes different tactics such as activating servers, enabling CAPTCHA, blacklisting clients, enabling authentication, or throttling suspicious clients.

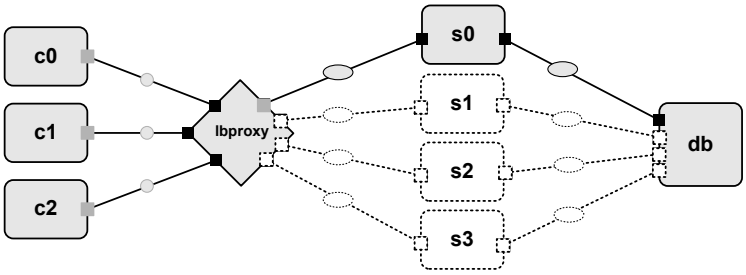


Figure 3.2: Znn.com architecture.

In Znn.com we can easily identify different types of uncertainties in the sensors and effectors of the system, summarized in Table 3.3, along with example locations in the system.

Uncertainty Source	Uncertainty Identified	Example Location in Znn.com
Sensor	Range	Server load
	Response Time	Server request service rate
	Error Distribution	Server Load / Server request service rate / Server byte service rate
	Expiration	
	Aggregate	
Effector	Success Rate	Activate server
	Execution Time	

Table 3.3: Uncertainties identified in Znn.com.

The range of the server load was not specified, so having set values would allow us to know when a sensor reading is unreasonable. The server request service rate should take some time since it would need to take a sample over time. The error distribution and aggregate uncertainties can be used in the Kalman filters in the gauges for noisy sensor readings. Also, the filters can be used to estimate the service rate based on the load, if the request rate information is not available. Finally, the uncertainty in activating a new server is specified. As mentioned before, activating a server may fail at times, and the execution time for the server to start up will take time.

3.4 Conclusion

This chapter has presented a method to represent different types of uncertainty in MAPE-K self-adaptive systems. By modeling uncertainties in Acme, we make available explicit specifications that can be employed by different activities in MAPE to mitigate certain kinds of uncertainty. Modeling explicitly this uncertainties entails a cost, both in terms of specification effort, and also information retrieval (which may not always be available, and/or may be costly to obtain).

However, factoring in uncertainty specifications for mitigating their potential adverse effects can pay off in many circumstances. In the next chapter, we describe how we can systematically reason about uncertainty specifications (in this case corresponding to sensors) to avoid incurring penalties when adapting software systems at run time.

Chapter 4

Reasoning about Uncertainty

Self-adaptive systems are expected to respond to unanticipated run-time events. This entails mitigating uncertainties, which are not always modeled as a first-class concern. In Chapter 3 we have seen how different types of uncertainty can be explicitly modeled in a MAPE-K system that employs architecture-centric models to describe the managed subsystem (Rainbow).

In this chapter, we employ some of the constructs for the explicit representation of uncertainties described in the previous chapter and use them to reason about decision-making in adaptation. Concretely, we focus on the *aleatoric* or *variable* uncertainty (i.e., caused by the randomness of events, rather than by lack of knowledge), and illustrate our approach on examples that deal with inaccuracies in sensing. We introduce a formal analysis technique based on model checking of stochastic multiplayer games (SMGs) that enables us to quantify the potential benefits of considering explicitly this type of uncertainty in decision making for adaptation.

The remainder of this chapter first presents a simple scenario to introduce the concepts and facilitate understanding of our technique (Section 4.1). Then, Section 4.2 describes our technique for analyzing adaptation based on model checking of stochastic games, experiment results, and observations.

4.1 A Simple Model

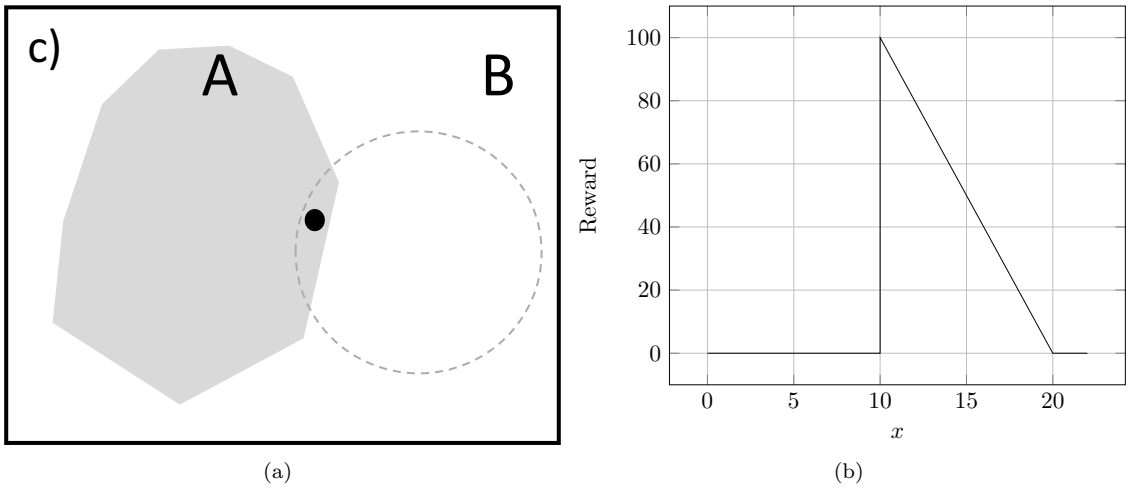
To motivate our approach, we describe a simple scenario illustrated in Figure 4.1(a), where the system/environment state space is divided in regions A and B. We assume that the sensors employed to monitor some of the variables that form the system/environment state are not very accurate, and therefore the monitoring infrastructure cannot determine the exact system/environment state.

Figure 4.1(b) introduces the concept of *reward*, which is an indicator of how well the system is meeting its goals (e.g. minimizing malicious users or maximizing requests served). Every time a system takes some action, it can collect certain reward based on how this action impacts the state of the system (and how well the new state aligns with system goals). The higher the reward, the better the decision is. In other words, we assume that the system's target in this scenario is to accumulate as much reward as possible over time by taking a series of actions.

In this simple model, we assume that reward is only associated with metric x, as shown in Figure 4.1(b). When x is below 10, there is no reward; when x is equal to 10, the reward is maximum, and the reward decreases as x moves away from 10. Thus, the entire state space of this model is effectively divided into two regions:

- **Region A**: $x < 10$
- **Region B**: $x \geq 10$

Suppose that this system that can only perform one action that decreases the value of x. Hence, when the system determines that the current state lies within region B (according to the *observed* value of x), it



(a) (b)

Figure 4.1: Simple model scenario.

should try to decrease the value of x to maximize reward, making it as close as possible to 10 (but without going below 10). However, the sensor that monitors the value of x is not very accurate and the system has to make the best possible decision under the uncertainty that arises due to the inaccuracy of the sensing process.

The remainder of the section describes our approach to analyzing uncertainty-aware self-adaptation in this simple model, based on model checking of SMG.

4.1.1 Formal Model Definition

The purpose of this model is comparing uncertainty-aware adaptation, i.e., decision-making that considers explicitly uncertainty information (in this case induced by inaccuracies in sensing), against uncertainty-agnostic adaptation that assumes that there is no uncertainty in the information it employs for decision-making.

The model encodes a game played by an environment and a system player. The model can be instantiated in two variants: one in which the system player is uncertainty-aware, and another in which the system is uncertainty-agnostic. The details of how these different variants are used are explained in Section 4.1.2.

Defining the Players

There are two players in this model: Environment (*env*) and System (*sys*). These two players take turns to take actions. As shown in Figure 4.1 , the turn is controlled by the global variable *turn*. There are two other global variables: *real_x* represents the real value of x at any given time, whereas *obs_x* represents the value of x observed by the system (i.e., the value communicated by the inaccurate sensor to the system).

```
1  player sys target_system,[act],sensor,[sense] endplayer
2  player env environment,[generate] endplayer
3
4  const ENV_TURN=1;
5  const SYS_TURN=2;
6
7  global turn:[ENV_TURN..SYS_TURN] init ENV_TURN; // Used to alternate between players
8
9  global obs_x:[0..20];
10 global real_x:[0..20];
```

Listing 4.1: Player definition.

The game is played in alternating turns by the system and the environment players. A typical cycle of the game works in the following way:

1. The environment generates the real value of x ($real\_x$ - see Listing 4.2), line 7).

```
1  const INIT_X;
2  const error;
3  const MAX_TURNS ;
4  module environment
5    t : [−1..MAX_TURNS] init 0;
6    // 1.Generate value of real x
7    [] (t>=0 & t<MAX_TURNS) & (turn=ENV_TURN) −> (t'=t+1) & (turn'=SYS_TURN) & (real_x'=INIT_X);
8  endmodule
```

Listing 4.2: Simple environment model definition.

2. the system senses the value obs_x (Listing 4.3, line 2). The uncertainty in the sensing process is modeled by a simple uniform probability distribution, in which there is 0.5 probability that the sensor reads the value accurately (i.e., $obs\_x = real\_x$), and 0.5 probability the reading exceeds the real value of x by a constant $error$ (i.e., $obs\_x = real\_x + error$).

```
1  module sensor
2          [sense] true −> 0.5:(obs_x' = real_x) + 0.5:(obs_x' = real_x+error>=0?real_x+error:0);
3  endmodule
```

Listing 4.3: Sensor definition.

3. After obtaining the observed value obs_x, the system (Listing 4.4, line 8) can choose to:

   (a) do nothing (line 14),

   (b) reduce the value of $real\_x$, subtracting the value of a variable s_step from $real\_x$ (lines 10-12). s_step is just the saturated value of a constant $step$ supplied as parameter to the model, which represents the maximum magnitude of the modification that the system's effector can carry out on the value of x. For example, if $step = 3$, s_step can take values in $\{1, 2, 3\}$.

```
1  const step;
2  formula s_step = obs_x−step >= 10 | obs_x < 10 ? step : obs_x − 10;
3
4  module target_system
5  expected_x:[0..20]init 0;
6  new_info:[0..1]init 0;
7      // 2. Sense
8          [sense](new_info=0)&(turn=SYS_TURN)−>(new_info'=1);
9      // 3.a. Act
10         [act] (new_info=1)&(turn=SYS_TURN)−> (real_x'=real_x−s_step>=0?real_x−s_step:0)
11                         &(expected_x'=obs_x−s_step>=0?obs_x−s_step:0)
12                         &(turn'=ENV_TURN)&(new_info'=0);
13     // 3.b. Do nothing
14         [] (new_info=1)&(turn=SYS_TURN) −> (expected_x'=obs_x)&(turn'=ENV_TURN)&(new_info'=0);
15 endmodule
```

Listing 4.4: Simple system model definition.

   Note that in the listing above, $expected\_x$ encodes the expected value of x from the perspective of the system after its turn is completed (the expected value of x is built on the value of obs_x).

The cycle repeats until the maximum number of turns played by the system and the environment is reached. However, we assume in the rest of the discussion a single-turn game for the sake of clarity (i.e., the game ends after the environment, and then the system play one turn each).

Collecting Reward

There are three types of rewards in this model. We use each one of them to emulate different types of adaptation (Listing 4.5):

1. rIU: reward if the system has the accurate information to make a decision, i.e., when system knows $real\_x$.

2. rEU: reward if the system can only sense obs_x and the system is unaware of the uncertainty, i.e., this is the expected reward when the system is uncertainty-agnostic and assumes that obs_x is an accurate reading.

3. rEU_uncertain: reward if the system can only see obs_x, and the system is aware of the uncertainty. In this case, the system knows that there is a 0.5 probability that obs_x is not accurate and it calculates the reward factoring in this probability.

```
1
2  formula rU = (real_x<10? 0:200−10∗real_x);
3  formula rEU = (expected_x<10? 0:200−10∗expected_x);
4  formula rEU_uncertain = 0.5∗rEU+0.5∗rU;
5
6  rewards "rU" // Real Instantaneous utility reward
7    (turn=ENV_TURN) & (t>=1) : rU;
8  endrewards
9
10 rewards "rEU" // Expected instantaneous utility reward (uncertainty−agnostic adaptation)
11    (turn=ENV_TURN) & (t>=1) : rEU;
12 endrewards
13
14 rewards "rEU_uncertain" // Expected instantaneous utility reward (uncertainty−aware adaptation)
15    (turn=ENV_TURN) & (t>=1) :rEU_uncertain;
16 endrewards
```

Listing 4.5: Simple model reward structure definition.

4.1.2 Experiments and Observation

To compare the uncertainty-aware vs. non-uncertainty-aware adaptation in the simple model, we use rPATL specifications that enable us to analyze:

1. R_{real}: The maximum utility that the system can obtain when it has the accurate information (when the system tries to maximize the reward based on $real\_x$). We can get this value by generating a strategy using the following property:

$$\langle\langle \mathsf{sys} \rangle\rangle \mathsf{R}^{\mathsf{rIU}}_{\mathsf{max}=?}[\mathsf{F}^{\mathsf{c}}\ \mathsf{t} = \mathsf{MAX\_TURNS}] \tag{4.1}$$

2. $R_{u-agnostic}$:The maximum utility that adaptation is able to obtain without factoring uncertainty. To obtain this value, we proceed in two steps:

   (a) First, we generate a strategy using the following property that quantifies the maximum expected accrued reward that the system "believes" it can guarantee based on its beliefs (there is no uncertainty in the expected value of x because the value of obs_x is accurate):

$$\langle\langle \mathsf{sys} \rangle\rangle \mathsf{R}^{\mathsf{rEU}}_{\mathsf{max}=?}[\mathsf{F}^{\mathsf{c}}\ \mathsf{t} = \mathsf{MAX\_TURNS}] \tag{4.2}$$

   (b) We verify property 4.1 under the generated strategy for property 4.2. This quantifies the real utility achieved (based on the value of $real\_x$), under the strategy generated based on the beliefs of the system (i.e., the value of x is obs_x, and it coincides with the real one).

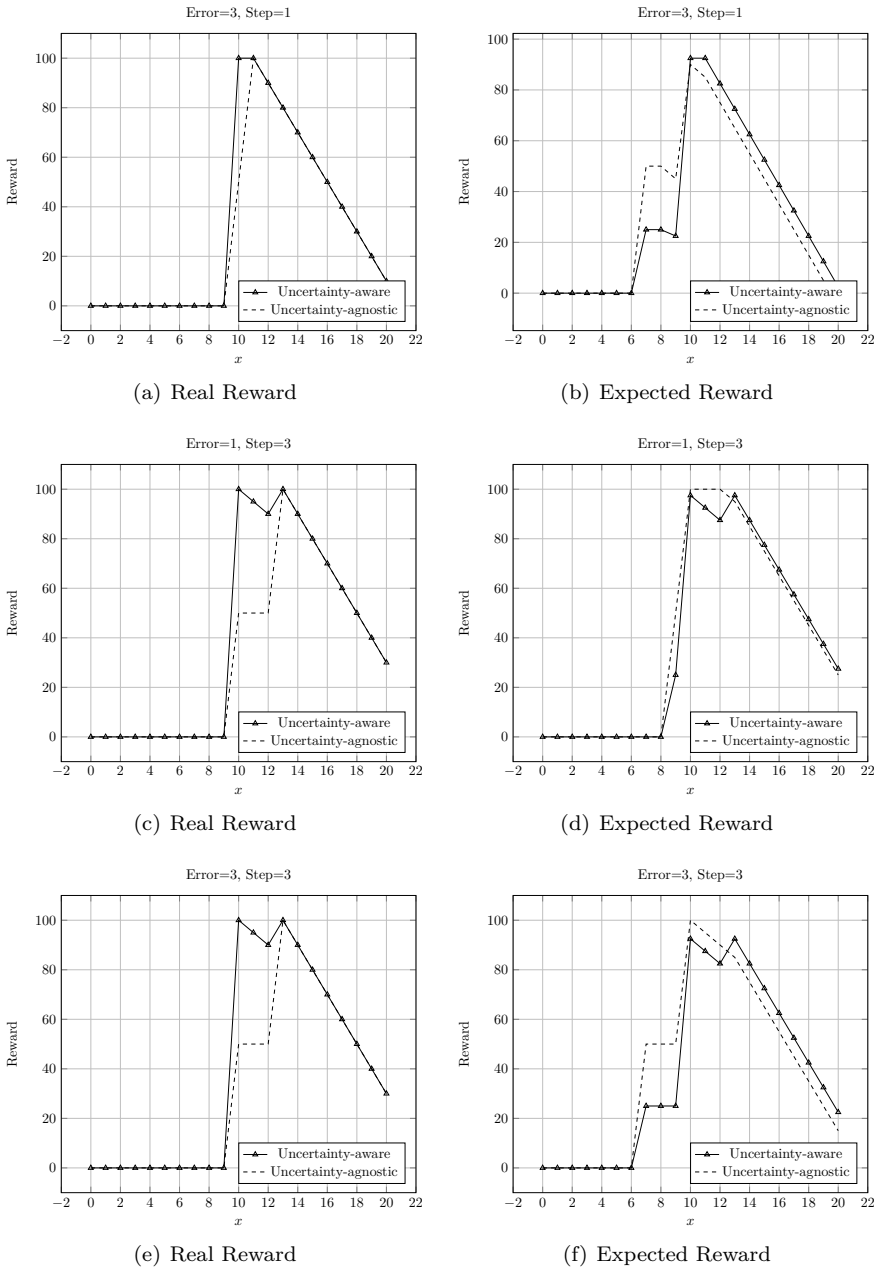
3. $R_{u-aware}$:The maximum utility that adaptation is able to obtain when considering uncertainty. To quantify this value, we proceed in two steps:

   (a) First, we generate a strategy using the following property that quantifies the maximum expected accrued reward that the system "believes" it can guarantee based on its beliefs. However, in this case the system is aware that the probability of $real\_x = obs\_x$ is only 0.5, so the strategy generated already accounts for the possibility of inaccurate readings. This is encoded in the reward $rEU\_uncertain$ in Listing 4.5 (line 4).

$$\langle\langle \mathsf{sys} \rangle\rangle \mathsf{R}^{\mathsf{rEU\_uncertain}}_{\mathsf{max}=?}[\mathsf{F}^{\mathsf{c}}\ \mathsf{t} = \mathsf{MAX\_TURNS}] \tag{4.3}$$

(b) We verify property 4.1 under the generated strategy for property 4.3. This quantifies the real reward under the strategy generated for uncertainty-aware decision-making.

For our experiment, we collected the value of reward for uncertainty-aware and uncertainty-agnostic adaptation in different cases, with sensor *error* and actuator impact *step* taking values in $\{1, 3\}$. The range of values for x explored is $\{0, \ldots, 20\}$. [1]



(a) Real Reward (b) Expected Reward

(c) Real Reward (d) Expected Reward

(e) Real Reward (f) Expected Reward

Figure 4.2: Simple model scenario results .

Figure 4.2 shows the results of our experiments. The plots on the left-hand side of the figure compare the reward obtained by uncertainty-aware and uncertainty-agnostic adaptation (i.e., $R_{u-aware}$ and $R_{u-agnostic}$,

[1] In this report, reference to the value of x without any further attribution indicates *real value of x*.

respectively), whereas the plots of the right-hand side show the expected reward of each one of the decision-making variants (i.e., quantified directly using properties 4.3 and 4.2 for uncertainty-aware, and uncertainty agnostic adaptation, respectively).

Looking at the results, we can make the following observations:

1. *When in a "safe" region, uncertainty does not matter.* When the value of x is in region A ($x \geq 10$), and not close to the threshold, the reward obtained is not affected by uncertainty in any way, since there is no risk that the system will modify the value below the threshold, leading to a loss of reward. In practice, both strategy-aware and agnostic adaptations will choose to reduce the value of x to obtain more reward. This can be observed in plots (a), (c), and (e) of Figure 4.2, in which the reward obtained by both adaptation variants converge as x moves to higher values, away from the threshold $x = 10$. Similarly, when the system is in region B ($x < 10$) uncertainty does not make any difference, since there is nothing that the system can do to collect more reward. So, both adaptation variants will behave in the same way.

2. *When close to the boundary between regions, uncertainty-aware adaptation performs better.* When the system is in region A, but in values that get close to the boundary between regions A and B, there is a chance that the system will make a sub-optimal decision due to the uncertainty in sensing. Concretely, in the uncertainty-agnostic variant of adaptation, the system can determine that it is safe to reduce the value of x by a given amount based by the value of obs_x (when in reality, the value of x will go below 10 and reward will not be collected). This penalizes uncertainty-agnostic adaptation with respect to the uncertainty-aware variant, which is already accounting for the likelihood of an undesirable outcome, and is more conservative when choosing to reduce the value of x. Plots (a), (c), and (e) of Figure 4.2 show how the different choices of adaptation variants lead to increased rewards in uncertainty-aware adaptation when the value of x is close to the boundary between regions. Moreover, Plots (b), (d), and (f) how uncertainty-agnostic adaptation tends to be more "optimistic" about the expected reward that will be obtained in the same areas in which uncertainty-aware adaptation performs better. In particular, we can observe if we focus on plot (b) that the range of values in which the real value of x is smaller than 10, but its observed value is 10 or higher (i.e., $7 \leq x < 10$), there is a bump in expected utility (which should be zero, if sensing was perfectly accurate).

3. *The difference between adaptation approaches is greater when sensor error is paired with actuator impact.* As sensor error increases, we would expect to see uncertainty-agnostic adaptation's reward progressively decrease. However, this is only true if sensor error is paired with higher actuator impact values, since otherwise the limited scope of the actuator mitigates the potentially detrimental effects that making the wrong choice would have on reward. For instance, if $error = 3$, but step=1, the plot in Figure 4.2 (a) shows that there is little performance difference between the two variants of adaptation. This is because, even if the system makes the wrong choice under uncertainty-agnostic adaptation, e.g., when $x = 12$, the actuator can at most reduce x to a value 11, incurring only a light penalty, compared to uncertainty-aware adaptation. However, if we consider the same value of $x = 12$ when $step = 3$, the difference in reward between decision-making approaches is much more pronounced because in situations in which reducing x is the wrong choice, it is more likely that x will go under the threshold $x = 10$, incurring a higher penalty.

4.2 A More Complex Model

In this section, we describe our approach on a more complex scenario, in which an enterprise web infrastructure similar to the Znn.com system described in Section 3.3 is experiencing a Denial of Service (DoS) attack.

When web infrastructure experiences unusually high traffic, one possibility is that it is under a DoS attack. However, it could also be a false positive when this traffic is caused by legitimate visitors (e.g., *slashdot effect*). Treating legitimate users as DoS attackers by mistake, applying strategies like blocking their requests for accessing the website could be harmful to the business. Thus, uncertainty about such situations should be

considered when applying defensive adaptation strategies to system, evaluating carefully the benefit and cost of different adaptation choices.

To facilitate the understanding of the DoS adaptation scenario, we structure our model in a similar way to the simple model described earlier in this chapter. Concretely, we make the following assumptions:

1. *The entire space is divided into two regions.* The state space is divided into two regions: (i) *DoS*, in which we assume that the system is experiencing an attack, and (ii) *Normal*, in which the system does not experience any anomalous activity that indicates a DoS attack.

2. *Regions are associated with specific metrics.* The system's state is determined by a single metric that captures the estimated *percentage of malicious clients* accessing the system (mc). This metric is an abstract concept that we use as proof of concept. In this scenario, we assume that if mc is above a given threshold, the system is in the DoS region of the state spece; otherwise the system is considered to be at the $Normal$ region.

3. *The system does not know the real value of metrics.* Just like in the simple model described in the previous section, the observable value of the metric mc may not reflect its real value. Hence, the system needs to make a decision using observable values only.

4. *Uncertainty in sensing is represented by probability distribution.* We use probability theory to represent uncertainty. Concretely, we employ a normal distribution function to model the observed percentage of malicious clientes mc.

5. *The uncertainty-aware version of the system has knowledge about the probability distribution function that captures uncertainty in sensing.* To simplify the problem, we currently assume that the system has knowledge of the probability distribution function that represents how observed values are generated during the sensing process. In the real world, this knowledge may be obtained for instance, from historical data. This issue is further discussed in Section 4.2.1.

The two main extensions with respect to the simple scenario presented earlier in this chapter are: (i) a richer set of actions or *tactics* that the system can carry out to influence state variables, and (ii) a more sophisticated notion of reward that factors in metrics along more than one dimension of concern.

- *Tactics.* In the simple model, the system can either do nothing or *act* on the value of x by decreasing it. In the real world, a system may have a richer variety of adaptation actions or *tactics* in terms of responding to run-time events and meet its goals. In this model, we divide tactics into two kinds:
  - *Uncertainty Reduction Tactics.* This kind of tactic can reduce uncertainty (in our scenario the uncertainty associated with the sensing of system metrics). This type of tactic often comes at a cost. For example, introducing captcha can reduce uncertainty about the maliciousness of clients accessing the system (by determining which ones are controlled by bots) and therefore, but it will increase the annoyance of legitimate users, who find their activities disrupted by the captcha.
  - *Non Uncertainty Reduction Tactics.* This kind of tactic does not reduce uncertainty. For example, *blackholing* clients (i.e., dropping their incoming requests) does not provide any new information about who is controlling the clients accessing the system. Decisions of whether to exercise these tactics are therefore highly depend on the quality of the information available to the system (the observed values of metrics).

- *Reward Model.* In the simple scenario, the reward is related only to one dimension (Figure 4.5). However, in this scenario there are two dimensions of concern: security and user experience, and we assume them to be of equal importance. Security is directly related to the metric percentage of malicious clients mc (lower is better). User experience is also affected by system's choice of applying different tactics. For example, introducing captcha will increase the difficulty of legitimate users accessing system services and therefore increase their annoyance. We consider therefore user annoyance (ua) as an additional metric for the user experience dimension of concern (lower is better).

4.2.1 Formal Model Definition

This section describes the implementation of our scenario in PRISM-games. Figure 4.3 describes how the model works. The scenario is modeled as a stochastic game involving two players that represent the system (*sys*) and the environment (*env*):

- The system player consists of two processes or *modules* that represent the *target system* and the *gauge* that collects observed values of the *mc* metric. These two modules are synchronized by the shared action *gauge*.
- The environment player consists of the *generator* and *environment* modules, which are synchronized via the *generateMc* shared action.

When the game starts, the environment player first generates the real value of the *mc* , and it yields the turn to the system player. Next, the system player gauges *mc*, producing its observed value. The system player then infers the region of the state space (Dos or Normal) based on the observed system metric , chooses one of the available tactics to execute (or chooses not doing anything), and returns the turn to the environment. The remainder of this section explains in more detail how each of the actions described in this loop is modeled in the stochastic game.

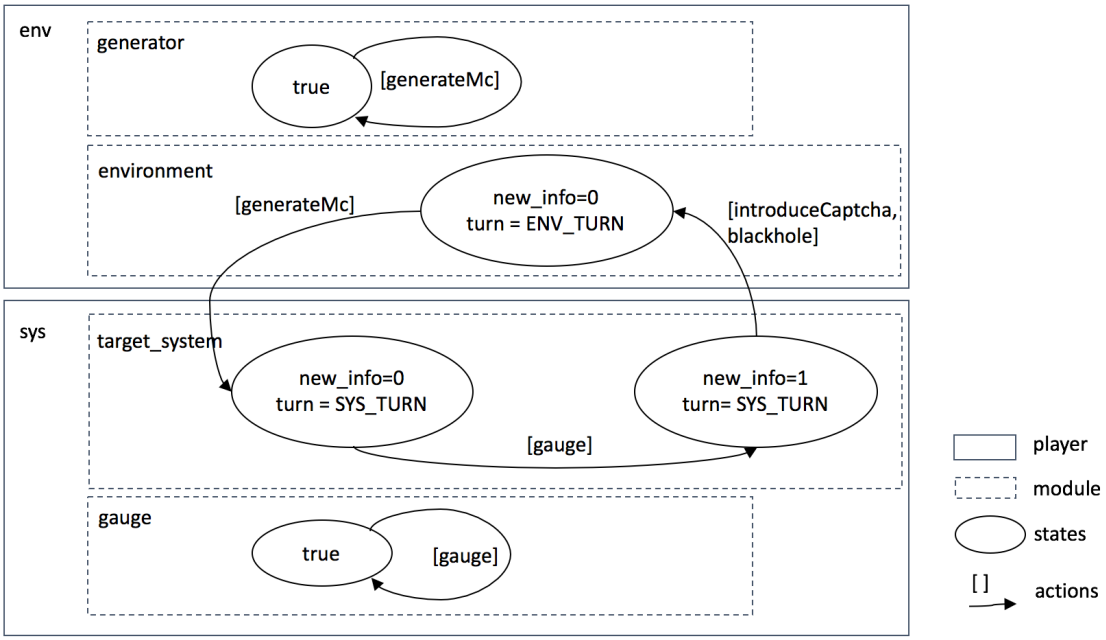


Figure 4.3: DoS scenario game model overview.

Defining the Players

The scenario is modeled as a stochastic game involving two players representing the system and the environment.

```
1 ENV_TURN=1;
2 const SYS_TURN=2;
3 global turn:[ENV_TURN..SYS_TURN] init ENV_TURN; // Used to alternate between players
4 global std_mc : [0..100]; //Observed standard deviation of malicious clients
5 global obs_mc:[0..100]; //observerd percentage of malicious clients
6 global real_mc:[0..100]; //real percentage of malicious clients
```

Listing 4.6: Player definition.

The formal model contains three global variables (Figure 4.6):

1. *real_mc*: The real percentage of malicious clients,ranging from [0,100], represents the real system metric.

2. *obs_mc*: The observed percentage of malicious clients.

3. *std_mc*: The standard deviation associated with the perceived percentage of malicious clients. Note that *obs_mc* and *std_mc* together describe the uncertainty function for the observed system metric.

Generating the Real Value of Metric

generator is a module that is responsible for generating the real value of metric (*real_mc*) during every turn of the environment player.

```
module generator
    [generateMc] true->(real_mc'=real_mc);
endmodule
```

Listing 4.7: Generator module definition.

Gauging Information

The *gauge* module is responsible for gauging information. This process is crucial to our scenario model because it encodes how observed values of *mc* are generated from the real values of the variable (i.e., it captures the source of aleatoric uncertainty in the sensing process). Concretely, the observed value of metric can be captured as the function:

$$P(x) = \frac{1}{std\_mc\sqrt{2\pi}} e^{-(x-obs\_mc)^2/2std\_mc^2}$$

or

$$f(x) = P[X = x]$$

This probability density function is actually a conditional probability distribution of the observed value of the metric, given a real value ($P(obs\_mc/real\_mc)$). In this scenario, we encode this function using six points to simulate this normal distribution (Listing 4.8).

```
module gauge
        [gauge]true->0.34:(obs_mc' = (real_mc+1*std_mc<=100?real_mc+1*std_mc:100))+
             0.34:(obs_mc' = (real_mc-1*std_mc>=0?real_mc-1*std_mc:0))+
         0.14:(obs_mc' = (real_mc+2*std_mc<=100?real_mc+2*std_mc:100))+
             0.14:(obs_mc' = (real_mc-2*std_mc>=0?real_mc-2*std_mc:0))+
                 0.02:(obs_mc' = (real_mc+3*std_mc<=100?real_mc+3*std_mc:100))+
             0.02:(obs_mc' = (real_mc-3*std_mc>=0?real_mc-3*std_mc:0));
endmodule
```

Listing 4.8: Gauge module definition.

Selecting Tactics

After the system obtains the information about system metrics, it can choose a tactic for execution (or do nothing). In this section, we study the performance of alternative selection strategies:

1. *Uncertainty-agnostic*. The system does not have knowledge about the real value of the metric *mc*. It also oblivious about the fact that there is uncertainty in the gauging process and therefore treats the observed value as the real information, selecting tactics based on this information.

2. *Uncertainty-aware*. The system does not have knowledge of the real value of *mc*. However, the system has knowledge about the uncertainty in the gauging process and evaluates the expected result considering the probability distribution over different system states and selects tactics based on that.

The adaptation decision is evaluated for the three selection strategies based on the value of the following set of variables:

1. *real_mc* . Real value of metric *mc*.
2. *emc*: This is the expected percentage of malicious client after executing a tactic, assuming that $obs\_mc = real\_mc$.
3. *ua*: This is the real value for the metric user annoyance.
4. *eua*: This is the expected user annoyance after executing a tactic assuming $obs\_mc = real\_mc$
5. *eua_dos*: This is the expected user annoyance after practicing a tactic if the system is currently at the DoS region.
6. *eua_normal*: This is the expected user annoyance after executing a tactic if the system is currently at Normal region. This variable and *eua_dos* are used to calculate the expected reward when the system is aware of the uncertainty.

These six variables are used to calculate three types of reward (real, expected by uncertainty-aware decision-making, and expected by uncertainty-agnostic decision-making). By maximizing different types of reward, we can employ our formal model to generate adaptation decisions for: (a) adaptation based on real information, (b) uncertainty-agnostic adaptation, and (c) uncertainty-aware adaptation. More details about the definition of these different types of reward are discussed in Section 4.2.1.

Executing Tactics

Table 4.1 summarizes the effect of exercising different tactics at different system states. For example, blackholing in both regions (DoS and Normal) reduces the *real_mc* by 30% whereas it increases user annoyance by 50% if the system is not under DoS, and by 10% if it is (reflecting the assumption that most clients will correspond to malicious users[2]).

real_mc	Normal State	DoS State
IntroduceCaptcha	-10	-10
Blackhole	-30	-30
ua	Normal State	DoS State
IntroduceCaptcha	+10	+10
Blackhole	+50	+10

Table 4.1: Simple impact specification of tactics in a DoS adaptation scenario.

Listing 4.9 shows how the tactic *blackhole* updates the six variables discussed in Section 4.2.1.

```
1  formula bh_f_mc = real_mc+bh_mc_delta >=0 ? (real_mc+bh_mc_delta<=100? real_mc+bh_mc_delta : 100) : 0;
2  formula bh_f_emc = obs_mc+bh_mc_delta >=0 ? (obs_mc+bh_mc_delta<=100? obs_mc+bh_mc_delta : 100) : 0;
3
4  formula bh_f_ua_normal = ua+50 >=0 ? (ua+50<=100? ua+50 : 100) : 0;
5  formula bh_f_ua_dos = ua+10 >=0 ? (ua+10<=100? ua+10 : 100) : 0;
6
7  formula bh_f_ua = (real_mc>dos_threshold)?bh_f_ua_dos:bh_f_ua_normal;
8  formula bh_f_eua = (obs_mc>dos_threshold)?bh_f_ua_dos:bh_f_ua_normal;
9  ...
10
11 module target_system
12 ...
13 [blackhole] (new_info=1)& (turn=SYS_TURN)-> (real_mc'=bh_f_mc) & (emc' = bh_f_emc)
14                          &(ua'=bh_f_ua)& (eua'=bh_f_eua)
15                          &(eua_dos'=bh_f_ua_dos)&(eua_normal'=bh_f_ua_normal)
16                          &(turn'=ENV_TURN)&(new_info'=0)&(executed'=2);
17 ...
18 endmodule
```

Listing 4.9: Blackhole tactic definition.

[2]In this model, we assume that the effect of tactics is deterministic

Collecting Reward

Rewards are calculated based on both user annoyance and the percentage of malicious clients. In this case, the reward we employ for our game encodes a simple utility function in which both metrics contribute to the overall utility calculation with a weight of 0.5. We employ three types of rewards to analyze uncertainty-aware and uncertainty-agnostic decision-making.

1. rIU (real utility): This reward is calculated based on the real value of the percentage of malicious clients ($real\_mc$).

2. rEIU (expected utility for uncertainty-agnostic decision making): Is calculated based on the observable information about the percentage of malicious clients (obs_mc), and it is unaware of the uncertainty in sensing.

3. rEIU_uncertain (expected utility for uncertainty-aware decision making): Is also calculated based on the observed value of the metric mc ((obs_mc). However, this alternative considers the uncertainty in sensing, since it draws the values for reward calculation based on all the possibilities captured in the six-point probability distribution encoded in Listing 4.10.

The calculation of $rEIU\_uncertain$ is the key of this model. When collecting reward in uncertainty-aware adaptation, we derive all the potential real values of metric mc, based on its observed value (Listing 4.11), and calculate the expected reward based on the probability distribution of these real values. In other words, **the system must have knowledge of the probability distribution of real values of the metric conditioned to its observed value P($real\_mc$ | obs_mc) to calculate** $rEIU\_uncertain$.

In the complex model, the observed value (obs_mc) is normally distributed given a real value ($real\_mc$) (Section 4.2.1). Based on the joint probability mass function of two discrete random variables:

$$P(X = x, Y = y) = P(Y = y \mid X = x) * P(X = x) = P(X = x \mid Y = y) * P(Y = y) \tag{4.4}$$

```
1  const double wA=0.5;
2  const double wM=0.5;
3
4  // Real Instantaneous utility reward (RGA)
5  rewards "rIU"
6    (turn=ENV_TURN) & (t>=1) : wM*uM + wA*uA;
7  endrewards
8
9  // Expected Instantaneous utility reward (RGA)
10 // unware of uncertainty
11 rewards "rEIU"
12    (turn=ENV_TURN) & (t>=1) : wM*EuM + wA*EuA;
13 endrewards
14
15 // Expected Instantaneous utility reward (RGA)
16 // aware of uncertainty
17 rewards "rEIU_uncertain"
18    (turn=ENV_TURN) & (t>=1) :0.34*(wM*EuM_pos_one+wA*ua_pos_one)+
19                    0.14*(wM*EuM_pos_two+wA*ua_pos_two)+
20                    0.02*(wM*EuM_pos_three+wA*ua_pos_three)+
21                        0.34*(wM*EuM_neg_one+wA*ua_neg_one)+
22                    0.14*(wM*EuM_neg_two+wA*ua_neg_two)+
23                    0.02*(wM*EuM_neg_three+wA*ua_neg_three);
```

Listing 4.10: Reward functions.

Listing 4.12 shows the utility profile for security, based on the values of the metric for client maliciousness. Moreover, Listing 4.11 shows the utility profile for user experience (calculated based on user annoyance ua) and security (calculated based on percentage of malicious client. Both profiles are defined using an encoding similar to the ones described in [SCG+14], where a detailed discussion of their rationale can be found.

```
1  //User Annoyance utility function
2  formula uA = (ua>=0 & ua <=100? 1-(ua/100):0);
3  formula EuA = (eua>=0 & eua <=100? 1-(eua/100):0);
4  formula EuA_normal = (eua_normal>=0 & eua_normal <=100? 1-(eua_normal/100):0);
5  formula EuA_dos = (eua_dos>=0 & eua_dos <=100? 1-(eua_dos/100):0);
```

```
6
7  //derive the 6 possible value of real_mc based on the obs_mc
8  formula real_pos_one_std = obs_mc +1*std_mc<=100?obs_mc +1*std_mc:100;
9  formula real_pos_two_std = obs_mc +2*std_mc<=100?obs_mc +2*std_mc:100;
10 formula real_pos_three_std = obs_mc +3*std_mc<=100?obs_mc +3*std_mc:100;
11 formula real_neg_one_std = obs_mc −1*std_mc>=0? obs_mc −1*std_mc:0;
12 formula real_neg_two_std = obs_mc −2*std_mc>=0? obs_mc −2*std_mc:0;
13 formula real_neg_three_std = obs_mc −3*std_mc>=0? obs_mc −3*std_mc:0;
14
15 //derive the 6 possible ua rewards after applying a tactic
16 formula ua_pos_one = real_pos_one_std>dos_threshold? EuA_dos:EuA_normal;
17 formula ua_pos_two = real_pos_two_std>dos_threshold? EuA_dos:EuA_normal;
18 formula ua_pos_three = real_pos_three_std>dos_threshold?EuA_dos:EuA_normal;
19 formula ua_neg_one = real_neg_one_std>dos_threshold? EuA_dos:EuA_normal;
20 formula ua_neg_two = real_neg_two_std>dos_threshold? EuA_dos:EuA_normal;
21 formula ua_neg_three = real_neg_three_std>dos_threshold?EuA_dos:EuA_normal;
```

Listing 4.11: User annoyance utility calculation.

```
1  // 'Maliciousness' utility function based on real_mc
2  formula uM = (real_mc>=0 & real_mc <=5? 1 : 0)
3          +(real_mc>5 & real_mc <=20? 1+(0.80−1)*((real_mc−5)/(20−5)) :0)
4          +(real_mc>20 & real_mc <=50? 0.80+(0.40−0.80)*((real_mc−20)/(50−20)) :0)
5          +(real_mc>50 & real_mc <=70? 0.40+(0.00−0.40)*((real_mc−50)/(70−50)) :0)
6          +(real_mc>70 ? 0:0);
7
8  // 'Maliciousness' utility function based on emc (without uncertainty awareness)
9  formula EuM = (emc>=0 & emc <=5? 1 : 0)
10         +(emc>5 & emc<=20? 1+(0.80−1)*((emc−5)/(20−5)) :0)
11         +(emc>20 & emc <=50? 0.80+(0.40−0.80)*((emc−20)/(50−20)) :0)
12         +(emc>50 & emc <=70? 0.40+(0.00−0.40)*((emc−50)/(70−50)) :0)
13         +(emc>70 ? 0:0);
14
15 //Derive 6 possible mc after applying a tactic
16 //this only holds when tactics increase/decrease real_mc by a constant number regardless of what state system is at
17 formula mc_delta = (executed=1? ic_mc_delta:0)+(executed=2? bh_mc_delta:0);
18 formula pos_one_std = real_pos_one_std + mc_delta <=100? (real_pos_one_std+mc_delta>=0?real_pos_one_std+mc_delta:0):100;
19 formula pos_two_std = real_pos_two_std + mc_delta <=100? (real_pos_two_std+mc_delta>=0?real_pos_two_std+mc_delta:0):100;
20 formula pos_three_std = real_pos_three_std+mc_delta <=100? (real_pos_three_std+mc_delta>=0?real_pos_three_std+mc_delta:0):100;
21 formula neg_one_std = real_neg_one_std + mc_delta >=0? real_neg_one_std+mc_delta:0;
22 formula neg_two_std = real_neg_two_std + mc_delta >=0? real_neg_two_std+mc_delta:0;
23 formula neg_three_std = real_neg_three_std + mc_delta>=0? real_neg_three_std + mc_delta:0;
24
25 formula EuM_pos_one = (pos_one_std>=0 & pos_one_std <=5? 1 : 0)
26         +(pos_one_std>5 & pos_one_std<=20? 1+(0.80−1)*((pos_one_std−5)/(20−5)) :0)
27         +(pos_one_std>20 & pos_one_std <=50? 0.80+(0.40−0.80)*((pos_one_std−20)/(50−20)) :0)
28         +(pos_one_std>50 & pos_one_std <=70? 0.40+(0.00−0.40)*((pos_one_std−50)/(70−50)) :0)
29         +(pos_one_std>70 ? 0:0);
30
31 formula EuM_pos_two = (pos_two_std>=0 & pos_two_std <=5? 1 : 0)
32         +(pos_two_std>5 & pos_two_std<=20? 1+(0.80−1)*((pos_two_std−5)/(20−5)) :0)
33         +(pos_two_std>20 & pos_two_std <=50? 0.80+(0.40−0.80)*((pos_two_std−20)/(50−20)) :0)
34         +(pos_two_std>50 & pos_two_std <=70? 0.40+(0.00−0.40)*((pos_two_std−50)/(70−50)) :0)
35         +(pos_two_std>70 ? 0:0);
36 ...
```

Listing 4.12: Client maliciousness utility calculation (excerpt).

4.2.2 Experiments and Observation

To compare the uncertainty-aware with uncertainty-agnostic adaptation, we use rPATL specifications that enable us to analyze:

1. R_{real}: The maximum utility that the system can obtain when it has the accurate information (when the system tries to maximize the reward based on $real\_mc$). We can obtain this value by generating a strategy using the following property:

$$\langle\langle \mathsf{sys} \rangle\rangle \mathsf{R}^{\mathsf{rIU}}_{\mathsf{max}=?}[\mathsf{F}^{\mathsf{c}}\ \mathsf{t} = \mathsf{MAX\_TURNS}] \tag{4.5}$$

2. $R_{u-agnostic}$:The maximum utility that adaptation is able to obtain without factoring uncertainty. To obtain this value, we proceed in two steps:

(a) First, we generate a strategy using the following property that quantifies the maximum expected accrued reward that the system "believes" it can guarantee based on its beliefs (there is no uncertainty in the expected value of *mc* and *ua* because the value of *obs_mc* is accurate):

$$\langle\langle \mathsf{sys} \rangle\rangle \mathsf{R}^{\mathsf{rEU}}_{\mathsf{max}=?}[\mathsf{F}^{\mathsf{c}}\ \mathsf{t} = \mathsf{MAX\_TURNS}] \tag{4.6}$$

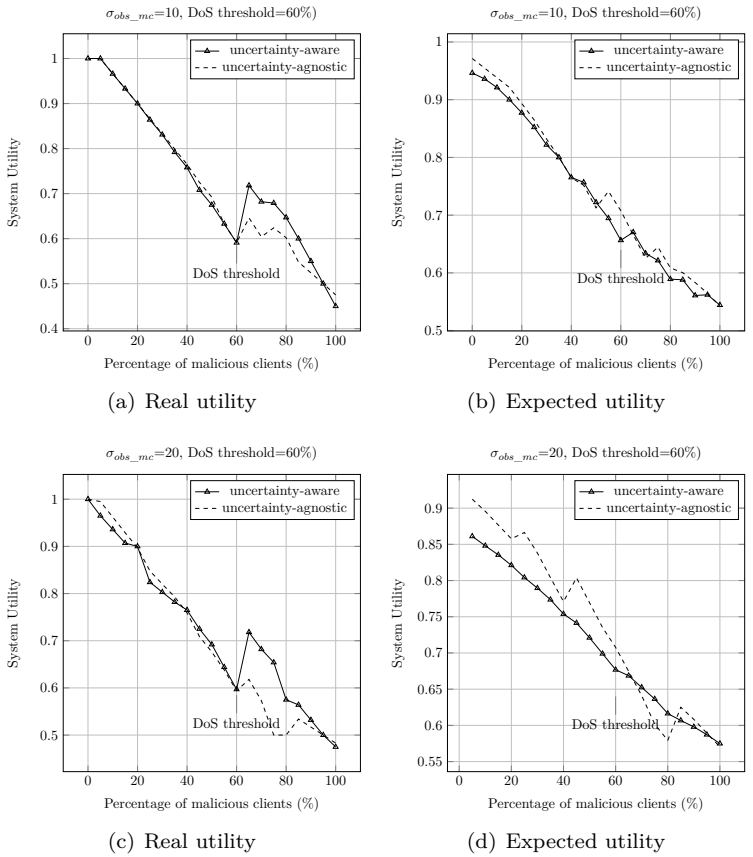
(b) We verify property 4.5 under the generated strategy for property 4.6. This quantifies the real utility achieved (based on the value of *real_mc*), under the strategy generated based on the beliefs of the system (i.e., the value of *mc* is *obs_mc*, and it coincides with the real one).

3. $R_{u-aware}$:The maximum utility that adaptation is able to obtain when considering uncertainty. To quantify this value, we proceed in two steps:

(a) First, we generate a strategy using the following property that quantifies the maximum expected accrued reward that the system "believes" it can guarantee based on its beliefs. However, in this case the system is aware that the probability of $real\_mc = obs\_mc$ depends on the distribution encoded in the gauge (Listing 4.8), so the strategy generated already accounts for the possibility of inaccurate readings.

$$\langle\langle \mathsf{sys} \rangle\rangle \mathsf{R}^{\mathsf{rEU\_uncertain}}_{\mathsf{max}=?}[\mathsf{F}^{\mathsf{c}}\ \mathsf{t} = \mathsf{MAX\_TURNS}] \tag{4.7}$$

(b) We verify property 4.5 under the generated strategy for property 4.7. This quantifies the real reward under the strategy generated for uncertainty-aware decision-making.



(a) Real utility (b) Expected utility

(c) Real utility (d) Expected utility

Figure 4.4: DoS scenario results.

For our experiment, we collected the value of reward for uncertainty-aware and uncertainty-agnostic adaptation in different cases, with a DoS threshold value of 60, and different sensor standard deviations in the distribution that captures the uncertainty in the sensing of mc $\sigma_{obs\_mc} \in \{10, 20\}$. The range of values for mc explored is $\{0, \ldots, 100\}$.

Figure 4.4 shows the result of our experiments, where expected utility results for both decision-making alternatives are shown on the right-hand side plots (b) and (d), and the real utilities obtained under the strategies generated for the expected utilities are shown on the left plots (a) and (c).

Some observations that can be drawn from the result of the experiments are the following:

1. *When far from region boundary, uncertainty does not matter.* When the value of mc is in region DoS ($mc \geq 60$), and moves away from the threshold, the utility obtained both by uncertainty-aware and agnostic adaptations is similar. This can be observed in plots (a) and (c) of Figure 4.4, in which the utility obtained by both adaptation variants converge as mc moves to higher values. Similarly, when the system is in region Normal ($mc < 60$) uncertainty only causes minor differences in performance between the two approaches.

2. *When close to the boundary between regions, uncertainty-aware adaptation performs better.* When the system is in region DoS, but in values that get close to the boundary between regions DoS and Normal, there is a chance that the system will make a sub-optimal decision due to the uncertainty in sensing. Concretely, in the uncertainty-agnostic variant of adaptation, the system can determine that it is safe to blackhole clients based by the value of *obs_mc* (when in reality, the value of mc will go below does not make this alternative that safe, and the system might incur in penalties for blackholing potentially legitimate clients). This penalizes uncertainty-agnostic adaptation with respect to the uncertainty-aware variant, which is already accounting for the likelihood of an undesirable outcome, and is more conservative when making a decision. Plots (a), (c) show how the different choices of adaptation variants lead to increased utility in uncertainty-aware adaptation when the value of mc is close to the boundary between regions.

3. *The difference between adaptation approaches is greater when standard deviation is higher.* As the standard deviation in sensor inaccuracies increases, we can see how the utility obtained by uncertainty-agnostic adaptation decreases. If we observe plots (a) and (c), focusing on the range in which percentage of malicious clients is between 60% and 90%, we can observe a noticeable drop in the utility obtained by the uncertainty-agnostic approach in plot (c), in which the standard deviation σ_{obs_mc} is doubled with respect to the one in plot (a).

4.3 Conclusion

This chapter has described an analysis technique based on model checking of stochastic multi-player games that enables us to quantify the benefits in adaptation performance of factoring uncertainty explicitly into decision-making. Our results show that although uncertainty-aware adaptation does not guarantee to perform better than non-uncertainty-aware adaptation in all cases, it does in most of the cases, and in particular, in the boundary of regions of the state space in which the dynamics of the system may change. This is a relevant finding, because systems that exhibit variability in the effects of adaptation tactics that depend on specific run-time conditions may obtain a remarkable benefit in terms of improved reliability and performance by factoring uncertainty into decision-making.

Chapter 5

Future Work

In this report, we have presented a method to represent different types of uncertainty in MAPE-K self-adaptive systems. By modeling uncertainties in Acme, we make available explicit specifications that can be employed by different activities in MAPE to mitigate certain kinds of uncertainty. Factoring in uncertainty specifications for mitigating their potential adverse effects can pay off in many circumstances.

However, modeling explicitly this uncertainties entails a cost, both in terms of specification effort, and also information retrieval (which may not always be available, and/or may be costly to obtain).

To mitigate this problem, a first research direction that we intend to pursue involves exploring alternatives to obtain actual values for uncertainty specifications. In particular, uncertainties related to error distribution in sensors or effector success rate may not have readily available values. One promising idea is extending the infrastructure in Rainbow to capture and modify those values in specifications and during run time, as the uncertainties change.

Another line of future work that we intend to explore is looking into alternative, more elaborate ways of specifying uncertainty. By using Acme, we were able to specify uncertainties in sensors, effectors, and human-in-the-loop. However, other uncertainties in goals and models, for example, are not represented. There has been a lot of research on uncertainty in goals and requirements specification, so integrating those with other notions of uncertainty in an architecture-based adaptation platform like Rainbow to study their interplay is an interesting research direction to explore.

In this report we have also described an analysis technique based on model checking of stochastic multi-player games that enables us to quantify the benefits in adaptation performance of factoring uncertainty explicitly into decision-making.

In this context, a first line of future work that we intend to explore entails analyzing the performance of uncertainty-aware adaptation when employing uncertainty-reduction adaptation tactics. One example in our DoS adaptation scenario is the tactic to enable captcha, which helps the system to better determine the percentage of malicious clients accessing the infrastructure. This tactic of course has a cost in terms of disrupting user activity, so exploring mechanisms that help the decision approach to determine when enacting uncertainty reduction tactics pays off is a natural extension to this work.

A second natural line of future work in this context concerns extending decision-making under uncertainty to reason only with partial knowledge about the uncertainty function. The work on decision-making described in this report assumes that the system has the knowledge of $P(observed\ \ value \mid real\ \ value)$ and can therefore derive $P(real\ \ value \mid observed\ \ value)$. A next logical step is to study how systems can gradually improve their ability of estimating the real state of the system, i.e. by automatically refining throughout subsequent system executions the knowledge that the system has about the $P(real\ \ value \mid observed\ \ value)$ function (assuming prior lack of knowledge of the function $P(observed\ \ value \mid real\ \ value)$).

Bibliography

[A+02] Rajeev Alur et al. Alternating-time temporal logic. *J. ACM*, 49(5), 2002.

[BdA95] Andrea Bianco and Luca de Alfaro. Model checking of probabalistic and nondeterministic systems. In *FSTTCS*, volume 1026 of *LNCS*. Springer, 1995.

[BPS10] L. Baresi, L. Pasquale, and P. Spoletini. Fuzzy goals for requirements-driven adaptation. In *Requirements Engineering Conference (RE), 2010 18th IEEE International*, pages 125–134, Sept 2010.

[C+11] Radu Calinescu et al. Dynamic QoS Management and Optimization in Service-Based Systems. *IEEE Trans. Software Eng.*, 37(3), 2011.

[CdLG+09] Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. Software engineering for self-adaptive systems: A research roadmap. In Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2009.

[CFK+13] Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, David Parker, and Aistis Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92, 2013.

[CFM13] Franck Chauvel, Nicolas Ferry, and Brice Morin. Models@runtime to support the iterative and continuous design of autonomic reasoners. In Nelly Bencomo, Robert B. France, Sebastian Götz, and Bernhard Rumpe, editors, *Proceedings of the 8th Workshop on Models @ Run.time co-located with 16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013), Miami, FL, USA, September 29, 2013.*, volume 1079 of *CEUR Workshop Proceedings*, pages 26–38. CEUR-WS.org, 2013.

[CG12] Shang-Wen Cheng and David Garlan. Stitch: A language for architecture-based self-adaptation. *Journal of Systems and Software*, 85(12):2860–2875, 2012.

[CGMS07] L. Cheung, L. Golubchik, N. Medvidovic, and G. Sukhatme. Identifying and addressing uncertainty in architecture-level software reliability modeling. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–6, March 2007.

[CGMS16] Javier Cámara, David Garlan, Gabriel A. Moreno, and Bradley Schmerl. *Evaluating Trade-Offs of Human Involvement in Self-Adaptive Systems.* Elsevier, September 2016.

[CGSA13] Paulo Casanova, David Garlan, Bradley Schmerl, and Rui Abreu. Diagnosing architectural runtime failures. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '13, pages 103–112, Piscataway, NJ, USA, 2013. IEEE Press.

[CK09] Radu Calinescu and Marta Z. Kwiatkowska. Using Quantitative Analysis to Implement Autonomic IT Systems. In *ICSE*, 2009.

[CKM15] Y. Chen, S. Kar, and J. M. F. Moura. Cyber-physical systems: Dynamic sensor attacks and strong observability. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1752–1756, April 2015.

[CL07] Taolue Chen and Jian Lu. Probabilistic alternating-time temporal logic and model checking algorithm. In *FSKD*, volume 2, 2007.

[CLGS14] Javier Cámara, Antónia Lopes, David Garlan, and Bradley Schmerl. Impact models for architecture-based self-adaptative systems. In *Proceedings of the 11th International Symposium on Formal Aspects of Component Software. FACS 2014, Bertinoro, Italy, September 10-12, 2014*, volume 8997 of *Lecture Notes in Computer Science*. Springer, 2014. To Appear. Available at: http://acme.able.cs.cmu.edu/pubs/show.php?id=421.

[CMG14] Javier Cámara, Gabriel A. Moreno, and David Garlan. Stochastic game analysis and latency awareness for proactive self-adaptation. In Gregor Engels and Nelly Bencomo, editors, *9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2014, Proceedings, Hyderabad, India, June 2-3, 2014*, pages 155–164. ACM, 2014.

[CMG15a] Javier Cámara, Gabriel A. Moreno, and David Garlan. Reasoning about human participation in self-adaptive systems. In Schmerl and Inverardi [SI15], pages 146–156.

[CMG15b] Javier Cámara, Gabriel A. Moreno, and David Garlan. Reasoning about human participation in self-adaptive systems. In Schmerl and Inverardi [SI15]. To appear.

[CNB+05] A. Cassandra, M. Nodine, S. Bondale, S. Ford, and D. Wells. Using pomdp-based state estimation to enhance agent system survivability. In *IEEE 2nd Symposium on Multi-Agent Security and Survivability, 2005.*, pages 11–20, Aug 2005.

[CSBW09] BettyH.C. Cheng, Pete Sawyer, Nelly Bencomo, and Jon Whittle. A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In *Model Driven Engineering Languages and Systems*, volume 5795 of *Lecture Notes in Computer Science*, pages 468–483. Springer Berlin Heidelberg, 2009.

[CWH15] M. S. Chong, M. Wakaiki, and J. P. Hespanha. Observability of linear systems under adversarial attacks. In *2015 American Control Conference (ACC)*, pages 2439–2444, July 2015.

[DKSS14] T. Deshpande, P. Katsaros, S.A. Smolka, and S.D. Stoller. Stochastic game-based analysis of the dns bandwidth amplification attack using probabilistic model checking. In *Dependable Computing Conference (EDCC), 2014 Tenth European*, pages 226–237, May 2014.

[E+09] Ilenia Epifani et al. Model Evolution by Run-Time Parameter Adaptation. In *ICSE*. IEEE CS, 2009.

[EEM10] Ahmed Elkhodary, Naeem Esfahani, and Sam Malek. Fusion: A framework for engineering self-tuning self-adaptive software systems. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '10, pages 7–16, New York, NY, USA, 2010. ACM.

[EGMT09] Ilenia Epifani, Carlo Ghezzi, Raffaela Mirandola, and Giordano Tamburrelli. Model evolution by run-time parameter adaptation. In Joanne M. Atlee and Paola Inverardi, editors, *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings*, pages 111–121. IEEE, 2009.

[EGT+09] G. Edwards, J. Garcia, H. Tajalli, D. Popescu, N. Medvidovic, G. Sukhatme, and B. Petrus. Architecture-driven self-adaptation and self-management in robotics systems. In *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 142–151, May 2009.

[EKM11] Naeem Esfahani, Ehsan Kouroshfar, and Sam Malek. Taming uncertainty in self-adaptive software. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 234–244. ACM, 2011.

[EM13] Naeem Esfahani and Sam Malek. Uncertainty in self-adaptive software systems. In Rogerio de Lemos, Holger Giese, Hausi Muller, and Mary Shaw, editors, *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *Lecture Notes in Computer Science*, pages 214–238. Springer, 2013.

[ES11] Douglas Eskins and William H. Sanders. The multiple-asymmetric-utility system model: A framework for modeling cyber-human systems. In *Eighth International Conference on Quantitative Evaluation of Systems, QEST 2011, Aachen, Germany, 5-8 September, 2011*, pages 233–242. IEEE Computer Society, 2011.

[ETAT15] M. Emami-Taba, M. Amoui, and L. Tahvildari. Strategy-aware mitigation using markov games for dynamic application-layer attacks. In *High Assurance Systems Engineering (HASE), 2015 IEEE 16th International Symposium on*, pages 134–141, Jan 2015.

[F+11] V. Forejt et al. Automated verification techniques for probabilistic systems. In *SFM*, volume 6659 of *LNCS*. Springer, 2011.

[FL03] Maria Fox and Derek Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *J. Artif. Int. Res.*, 20(1):61–124, December 2003.

[FTD14] H. Fawzi, P. Tabuada, and S. Diggavi. Secure estimation and control for cyber-physical systems under adversarial attacks. *IEEE Transactions on Automatic Control*, 59(6):1454–1467, June 2014.

[Gar10] David Garlan. Software engineering in an uncertain world. In *Proceedings of the Workshop on Future of Software Engineering Research, FoSER 2010, at the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, Santa Fe, NM, USA, November 7-11, 2010*, pages 125–128, 2010.

[GCH+04] David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley R. Schmerl, and Peter Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *IEEE Computer*, 37(10):46–54, 2004.

[GMS13] Carlo Ghezzi and Amir Molzam Sharifloo. *Dealing with Non-Functional Requirements for Adaptive Systems via Dynamic Software Product-Lines*, pages 191–213. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[GMW97] David Garlan, Robert T. Monroe, and David Wile. Acme: an architecture description interchange language. In J. Howard Johnson, editor, *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative Research, November 10-13, 1997, Toronto, Ontario, Canada*, page 7. IBM, 1997.

[GPST13] Carlo Ghezzi, Leandro Sales Pinto, Paola Spoletini, and Giordano Tamburrelli. Managing non-functional uncertainty via model-driven adaptivity. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 33–42, Piscataway, NJ, USA, 2013. IEEE Press.

[HM08] Markus C. Huebscher and Julie A. McCann. A survey of autonomic computing - degrees, models, and applications. *ACM Comput. Surv.*, 40(3), 2008.

[HSF04] Svein Hallsteinsen, Erlend Stav, and Jacqueline Floch. Self-adaptation for everyday systems. In *Proceedings of the 1st ACM SIGSOFT Workshop on Self-managed Systems*, WOSS '04, pages 69–74, New York, NY, USA, 2004. ACM.

[HZHC15] Kangli He, Min Zhang, Jia He, and Yixiang Chen. Probabilistic model checking of pipe protocol. In *Theoretical Aspects of Software Engineering (TASE), 2015 International Symposium on*, pages 135–138, Sept 2015.

[KC03] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36, 2003.

[KIL17] Gabriele Kern-Isberner and Thomas Lukasiewicz. Many facets of reasoning under uncertainty, inconsistency, vagueness, and preferences: A brief survey. *KI - Künstliche Intelligenz*, pages 1–5, 2017.

[KNP09] Marta Kwiatkowska, Gethin Norman, and David Parker. Stochastic games for verification of probabilistic timed automata. In Joel Ouaknine and Frits W. Vaandrager, editors, *Formal Modeling and Analysis of Timed Systems*, volume 5813 of *Lecture Notes in Computer Science*, pages 212–227. Springer Berlin Heidelberg, 2009.

[Kol56] A. N. Kolmogorov. *Foundations of the Theory of Probability*. Chelsea, New York, 1956.

[LSSS14] Wenchao Li, Dorsa Sadigh, S.Shankar Sastry, and SanjitA. Seshia. Synthesis for human-in-the-loop control systems. In Erika Abraham and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 8413 of *Lecture Notes in Computer Science*, pages 470–484. Springer Berlin Heidelberg, 2014.

[MHAW16] S. Mahdavi-Hezavehi, P. Avgeriou, and D. Weyns. A classification of current architecture-based approaches tackling uncertainty in self-adaptive systems with multiple requirements, 2016.

[Mye91] Roger B. Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, Cambridge, MA, 1991.

[PPM14] Diego Perez-Palacin and Raffaela Mirandola. Uncertainties in the modeling of self-adaptive systems: A taxonomy and an example of availability evaluation. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, ICPE '14, pages 3–14, New York, NY, USA, 2014. ACM.

[RJC12] Andres J. Ramirez, Adam C. Jensen, and Betty H. C. Cheng. A taxonomy of uncertainty for dynamically adaptive systems. In Hausi A. Müller and Luciano Baresi, editors, *7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2012, Zurich, Switzerland, June 4-5, 2012*, pages 99–108. IEEE Computer Society, 2012.

[RRdN+15] Ivan Ruchkin, Ashwini Rao, Dionisio de Niz, Sagar Chaki, and David Garlan. Eliminating inter-domain vulnerabilities in cyber-physical systems: An analysis contracts approach. In *Proceedings of the First ACM Workshop on Cyber-Physical Systems-Security and/or PrivaCy, CPS-SPC 2015*, pages 11–22, 2015.

[SCG+14] Bradley R. Schmerl, Javier Cámara, Jeffrey Gennari, David Garlan, Paulo Casanova, Gabriel A. Moreno, Thomas J. Glazier, and Jeffrey M. Barnes. Architecture-based self-protection: composing and reasoning about denial-of-service mitigations. In Laurie A. Williams, David M. Nicol, and Munindar P. Singh, editors, *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security, HotSoS 2014, Raleigh, NC, USA, April 08 - 09, 2014*, page 2. ACM, 2014.

[SI15] Bradley Schmerl and Paola Inverardi, editors. *10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015, Proceedings, Florence, Italy, May 18-19, 2015*. ACM, 2015.

[TVM+13] G. Tamura, N. M. Villegas, H. A. Muller, L. Duchien, and L. Seinturier. Improving context-awareness in self-adaptation using the dynamico reference model. In *2013 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 153–162, May 2013.

[VG10] Thomas Vogel and Holger Giese. Adaptation and abstract runtime models. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '10, pages 39–48, New York, NY, USA, 2010. ACM.

[VTM+13] Norha M. Villegas, Gabriel Tamura, Hausi A. Müller, Laurence Duchien, and Rubby Casallas. *DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems*, pages 265–293. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[WB95] Greg Welch and Gary Bishop. An introduction to the kalman filter. Technical report, Chapel Hill, NC, USA, 1995.

[WMA10] Danny Weyns, Sam Malek, and Jesper Andersson. On decentralized self-adaptation: Lessons from the trenches and challenges for the future. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '10, pages 84–93, New York, NY, USA, 2010. ACM.

[WSB+09] J. Whittle, P. Sawyer, N. Bencomo, B.H.C. Cheng, and J. Bruel. Relax: Incorporating uncertainty into the specification of self-adaptive systems. In *Requirements Engineering Conference, 2009. RE '09. 17th IEEE International*, pages 79–88, Aug 2009.

[Zad65] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338 – 353, 1965.

[Zad99] L.A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 100, Supplement 1(0):9 – 34, 1999.

[ZSA+16] Man Zhang, Bran Selic, Shaukat Ali, Tao Yue, Oscar Okariz, and Roland Norgren. *Understanding Uncertainty in Cyber-Physical Systems: A Conceptual Model*, pages 247–264. Springer International Publishing, Cham, 2016.

[ZSL14] Parisa Zoghi, Mark Shtern, and Marin Litoiu. Designing search based adaptive systems: A quantitative approach. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS 2014, pages 7–16, New York, NY, USA, 2014. ACM.

List of Figures

List of Tables