Analyzing Self-adaptation Via Model Checking of Stochastic Games

Javier Cámara[1], David Garlan[1], Gabriel Moreno[2], and Bradley Schmerl[1]

[1] ISR - Institute for Software Research
Carnegie Mellon University, Pittsburgh, PA 15213, USA
{jcmoreno,garlan,schmerl}@cs.cmu.edu
[2] SEI - Software Engineering Institute
Carnegie Mellon University, Pittsburgh, PA 15213, USA
gmoreno@sei.cmu.edu

Abstract. Design decisions made during early development stages of self-adaptive systems tend to have a significant impact upon system properties (e.g., safety, QoS) at runtime. However, understanding the outcome of these decisions *a priori* is difficult due to the different types and degrees of uncertainty that affect such systems (e.g., simplifying assumptions, human-in-the-loop). To provide some assurances about self-adaptive system designs, evidence can be gathered from activities such as simulations and prototyping, but these demand a significant effort and do not provide a systematic way of dealing with uncertainty. In this chapter, we describe an approach based on model checking of stochastic multiplayer games (SMGs) that enables developers to approximate the behavioral envelope of a self-adaptive system by analyzing best- and worst-case scenarios of alternative designs for self-adaptation mechanisms. Compared to other sources of evidence, such as simulations or prototypes, our approach provides developers with a preliminary understanding of adaptation behavior with less effort, and without the need to have any specific adaptation algorithms or infrastructure in place. We illustrate our approach by showing how it can be used to mitigate different types of uncertainty in contexts such as self-protecting systems, proactive latency-aware adaptation, and human-in-the-loop.

Keywords: Self-adaptation, Stochastic multiplayer games, Probabilistic Model Checking, Design-time assurances

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 [13, 28] is regarded as a promising way to engineer in an effective manner systems that are *resilient* to runtime changes in their execution environment (e.g., resource availability), goals, or even in the system itself (e.g., faults).

Self-adaptive approaches typically focus on enforcing safety and liveness properties [4, 26] during operation, and/or optimize qualities such as performance, reliability, or cost [25, 43]. However, providing actual *assurances* about the satisfaction of properties in a self-adaptive system is not easy in general, since its runtime behavior is largely influenced by the unpredictable behavior of the execution environment under which it is placed [22]. This is particularly true during the early stages of the development process, in which major design decisions that can have a large impact on the properties of the resulting system are made.

When developers face the construction of a self-adaptive system, there is a plethora of considerations that need to be made upfront, such as what the trade-offs between reactive and proactive adaptation are; whether decision-making should be centralized or decentralized; or how the concurrent execution of adaptations affects the performance and the reliability of a system, compared to a sequential execution model.

In general, the answer to these questions will be to a great extent informed by prior experience with similar existing systems, or by activities involving prototyping o simulation. However, experience with similar existing systems is not always available, and simulation and prototyping of (potentially many) system design variants is not cost-effective and does not provide systematic support analyze the system in the context of an unpredictable environment.

Ideally, developers should be able to use techniques that provide some feedback about the potential outcomes of early design choices in the development process, helping them to narrow down the solution space in a cost-effective manner.

In this chapter, we propose an approach to analyze self-adaptive systems while accounting explicitly for the uncertainty in their operating environment, given some assumptions about its behavior. The approach enables developers to approximate the behavioral envelope of a self-adaptive system by analyzing best- and worst-case scenarios of alternative designs for self-adaptation mechanisms. The formal underpinnings of our proposal are based on model checking of stochastic multiplayer games (SMGs), which is a technique particularly suited to analyzing the interplay of a self-adaptive system and its environment, since SMG models 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).

The main idea behind the approach is modeling the system and its environment as players in a SMG that can either cooperate to achieve a common goal (for best-case scenario analysis), or compete against each other (for worst-case scenario analysis). The approach is purely declarative, employing adaptation knowledge based on architectural

system descriptions as the scaffolding on which game model specifications are built. The analysis of such SMG specifications enables developers to obtain a preliminary understanding of adaptation behavior, and it can be used complementarily to other sources of evidence that typically require the availability of specific adaptation algorithms and infrastructure, such as prototypes or simulations.

In [8] we reported on a concrete application of this technique to quantify the benefits of employing information about the latency of tactics in proactive adaptation, comparing it against approaches that made the simplifying assumption of no tactic latency. In this chapter, we generalize our approach and show its versatility by describing how it can be instanced for new applications to deal with other sources of uncertainty (due to parameters over time and human-in-the-loop).

In the remainder of this chapter, Section 2 introduces some background and related work on different theories and approaches to support the construction of self-adaptive systems. Section 3 provides a general description of our approach, and Section 4 illustrates its application in different contexts, including the analysis of self-protecting systems, proactive latency-aware adaptation, and human-in-the-loop adaptation. Finally, Section 5 draws some conclusions and indicates directions for future work.

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 [30], fuzzy sets and possibility [41, 42], or games [35]) to factor in and mitigate the various types of uncertainty that affect self-adaptive systems.

Moreover, recent advances in formal verification [33, 11] have also explored the combination of probability and game theory to analyze systems in which uncertainty and competitive behavior are first-order elements.

In this section, we provide an overview of how different theories have been employed to analyze 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 [22] (Table 1).

2.1 Fuzzy Sets and Possibility Theory

Fuzzy set theory extends classical set theory with the concept of *degree of membership* [41], 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 [42] is based on fuzzy sets, and in its basic interpretation, it assumes that given a finite set (e.g., describing possible future

[3] 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 [38]						✓			
	FLAGS [2]						✓			
	POISED [21]	✓		✓[3]			✓			
Probability Theory	Rainbow [25]	✓		✓						
	FUSION [17]	✓	✓						✓	
	Calinescu and Kwiatkowska [6]				✓					
	KAMI [20]				✓					
	QoSMOS [5]				✓					
	MAUS [23]					✓				
Probability + Game Theory	Li et al. [34]					✓				
	Emami-Taba et al. [18]				✓					
	Cámara et al. [8, 9]	✓			✓	✓				

Table 1. Theories and approaches to mitigate uncertainty

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 [2, 21, 38]. RELAX [38] 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 runtime, depending on the state of the environment. FLAGS [2] 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 [21] 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.

2.2 Probability Theory

Probability theory [30] 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) [17, 25]. FUSION [17] 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 and the effects and system qualities, helping to mitigate uncertainty by considering explicitly interactions between adaptations. Rainbow [25] 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 [29]. 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 [6, 5, 19]. Calinescu and Kwiatkowska [6] 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 [19], a methodology and framework to keep models alive by feeding them with runtime data that updates their internal parameters. The framework focuses on reliability and performance, and uses Discrete-Time Markov Chains (DTMCs) and Queuing Networks as models to reason about the evolution of non-functional properties over time. Moreover, the QoSMOS framework [5] extends and combines [6] and [19] 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 [23] 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 Probability and Game Theory

Game theory is the study of mathematical models of conflict and cooperation between intelligent rational decision-makers [35]. 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 [34] 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. [18] 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.

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 [31] and communication protocols [27]. 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. 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 [11]. 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.

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 sets 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.[4]

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 [12] (a probabilistic version of ATL [1], 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 [3]. Moreover, rPATL includes a generalization of the reward operator $\mathsf{R}^{\mathsf{r}}_{\bowtie \mathsf{x}}$ from [24] to reason about goals related to rewards. An extended version of the rPATL reward operator $\langle\langle C \rangle\rangle \mathsf{R}^{\mathsf{r}}_{\mathsf{max}=?}[\mathsf{F}^{\star}\ \phi]$ [5] 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

[4] See Appendix A.2 in [11] for details.

[5] 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.

reward accumulated along paths leading to an end state that a player sys can guarantee, regardless of the strategies of other players."

Previous Work. We presented in [8] 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 [9] 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 [15] with OWC constructs [23].

3 Approach

This section describes our approach to analyzing self-adaptive systems via model checking of SMGs. The approach enables developers to approximate the behavioral envelope of a self-adaptive system operating in an arbitrary environment, on which some assumptions are made. Figure 1 illustrates the underlying idea behind the approach, which consists in modeling both the self-adaptive system and its environment as two players of a SMG. The system's player objective is optimizing an objective function encoded in a rPATL specification (e.g., minimizing the probability of violating a safety property, or maximizing accrued utility - encoded as a reward structure on the game). In contrast, the environment can either be considered as adversarial to the system (enabling worst-case scenario analysis), or as a cooperative player that helps the system to optimize its objective function (enabling best-case scenario analysis).

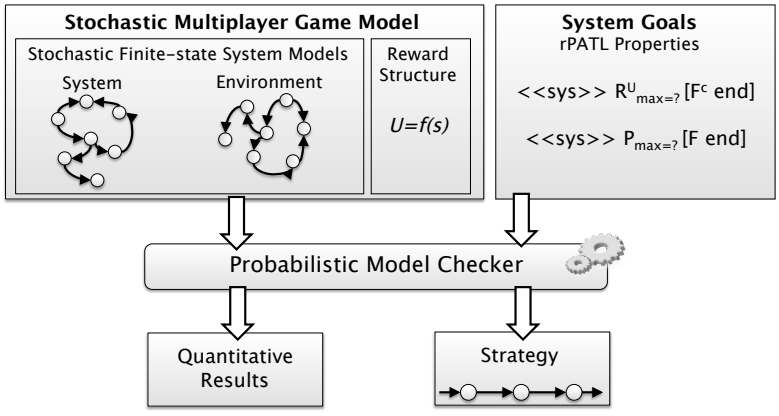

Fig. 1. Model checking of self-adaptation SMGs.

Our approach consists of two phases: (i) model specification, consisting in building the game model that describes the possible interactions between the self-adaptive system and its environment, and (ii) strategy synthesis, in which a game strategy that optimizes the objective function of the system player is built, enabling developers to quantify the outcome of adaptation in boundary cases.

3.1 Model Specification

Model specification involves constructing the solution space for the system by specifying a stochastic multiplayer game $\mathcal{G} = \langle \Pi, S, A, (S_i)_{i \in \Pi}, \Delta, AP, \chi, r \rangle$, where:

- $\Pi = \{sys, env\}$ is the set of players formed by the self-adaptive system and its environment.
- $S = S_{sys} \cup S_{env}$ is the set of states, where S_{sys} and S_{env} are the states controlled by the system and the environment players, respectively ($S_{sys} \cap S_{env} = \emptyset$). States are tuples of values for state variables that capture system and environment state properties represented in architecture models. Moreover, a special state variable t encodes explicitly in the game whether the current state belongs either to S_{sys} or S_{env}.
- $A = A_{sys} \cup A_{env}$ is the set of available actions in the game, where A_{sys} and A_{env} are the actions available to the system and the environment players, respectively.
- AP is a subset of all the predicates that can be built over the state variables. AP always includes a special predicate end, which labels explicitly all absorbing states of the model (for a state $s \in S$, we say that $s \models end$ iff $\forall s' \in S, a \in A \bullet \Delta(s, a)(s') \neq 0 \Rightarrow s = s'$). In practice, the end predicate abstracts the conditions leading to the end of the game (e.g., stop condition for the execution of the system).
- r is a reward structure labeling states with an associated cost or benefit. In the context of this chapter we assume a reward structure encoding utility. Specifically, the reward of an arbitrary game state s is defined as:

$$r(s) = \sum_{i=1}^{q} w_i \cdot u_i(v_i^s)$$

  where u_i is the utility function for quality dimension $i \in \{1, \ldots, q\}$, $w_i \in [0, 1]$ is the weight assigned to the utility of dimension i, and v_i^s denotes the value that quantifies quality attribute i in state s.

The state space and behaviors of the game are generated by performing the alphabetized parallel composition of a set of stochastic processes under the control of the system and environment players in the game (i.e., processes synchronize only on actions appearing in more than one process):

System Player The self-adaptive system (player sys) controls the choices made by two different processes:

- The *controller*, which corresponds to a specification of the behavior followed by the adaptation layer of the self-adaptive system, and can trigger the execution of tactics

upon the target system under adaptation. The set of actions available to the controller process A_{sys} corresponds to the set of available tactics in the adaptation model. Each of these actions $a \in A_{sys}$ is encoded in a command of the form:[6]

$$[a]\ C_a \wedge \neg end \wedge t = sys \rightarrow t' = env$$

Where the guard includes: (i) the conjunction of architectural constraints that limit the applicability of tactic a (abstracted by C_a, e.g., a new server cannot be activated if all of them are already active), (ii) a predicate $\neg end$ to avoid expanding the state space beyond the stop condition for the game, and (iii) a predicate to constrain the execution of actions of player sys to states $s \in S_{sys}$ (control of player turns is made explicit by a variable t that can take a value associated with any of the two players).

Note that in the most general case, all the local choices regarding the execution of controller actions are specified nondeterministically in the process, since this will enable the unfolding of all the potential adaptation executions when performing the parallel composition of the different processes in the model. However, the behavior of the controller can be further constrained to represent more specific instances of adaptation logic (e.g., as expressed in Stitch strategies) by including additional elements in the specification of the controller process. We illustrate both cases in the applications described in Section 4.

– The *target system*, whose set of available actions is also A_{sys}. Action executions in the target system synchronize with those in the controller process on the same action names. In this case, each action can be encoded in one or more commands of the form:

$$[a]\ pre_a \rightarrow p_a^1 : post_a^1 + \ldots + p_a^n : post_a^n$$
$$\ldots$$

Hence, a specific action in the controller can synchronize with any of the alternative executions of the same action in the target system. This models the different execution instances that the same tactic can have upon the target system (e.g., when the controller enlists a server, the target system can activate any of the alternative available servers). Each one of these commands is guarded by the precondition of a tactic's execution instance, denoted by pre_a (e.g., a specific server needs to be disabled in order to be enlisted). Moreover, the execution of a command can result in a number of alternative updates on state variables (along with their assigned probabilities p_a^i) that correspond to the different probabilistic outcomes of a given tactic execution instance, denoted by $post_a^i$ (e.g., the activation of a specific server can result in a successful boot with probability p, and fail with probability $1 - p$).

[6] We illustrate our approach to modeling the SMG using the syntax of the PRISM language [32] for Markov Decision Processes (MDPs), which are encoded as a set of commands of the form:

$$[action]\ guard \rightarrow p_1 : u_1 + \ldots + p_n : u_n$$

Where $guard$ is a predicate over variables in the model. Each update u_i describes a transition that the process can make (by executing $action$) if the guard is true. An update is specified by giving the new values of the variables, and has an assigned probability $p_i \in [0, 1]$. Multiple commands with overlapping guards (and commonly, including a single update of unspecified probability) introduce local nondeterminism.

Environment Player The environment (player *env*) controls one or more stochastic processes that model potential disturbances in the execution context out of the system's control such as network latency, or workload fluctuations. Each environment process is specified as a set of commands with asynchronous actions $a \in A_{env}$, and, similarly to the controller process, its local choices are specified nondeterministically to allow a broad range of environment behaviors within its possibilities. Each one of the commands follows the pattern:

$$[a]\ C_a^e \wedge \neg end \wedge t = env \rightarrow p_a^1 : post_a^1 \wedge t' = sys + \ldots + p_a^n : post_a^n \wedge t' = sys$$

Where C_a^e abstracts the set of environment constraints for the execution of action a (e.g., a threshold for the maximum latency that can be introduced in the network), and $\neg end$ prevents the generation of further states for the game. The command includes one or more updates, along with their associated probabilities. Each alternative update corresponds to one probabilistic outcome of the execution of a ($post_a^i$), and yields the turn to the system player.

3.2 Strategy Synthesis

Consists of generating a memoryless, deterministic strategy in $\mathcal{G}$ for player sys that optimizes its objective function. The specification of the objective for the synthesis of such strategy is given as a rPATL property that uses the operators $\mathsf{P}_{\bowtie \mathsf{q}}$ or $\mathsf{R}^{\mathsf{r}}_{\bowtie \mathsf{x}}$ to optimize probabilities or rewards, respectively.

- *Probability-based properties* are useful to maximize the probability of satisfying a given property (or conversely, minimize the probability of property violation). We consider properties encoded following the pattern:

  $$\langle\langle sys \rangle\rangle \mathsf{P}_{\bowtie \in \{max=?, min=?\}}[\mathsf{F}\ \phi]$$

  An example of such property would be $\langle\langle sys \rangle\rangle \mathsf{P}_{max=?}[\mathsf{F}\ \mathsf{success}]$, meaning "value of the maximum probability to reach a success state that the system player can guarantee, regardless of the strategy followed by the environment."
- *Reward-based properties* can help to maximize the benefit obtained from operating the system (e.g., in terms of utility), or minimize some cost (e.g., minimize the time to achieve a particular outcome when adapting the system). We consider properties encoded using the pattern:

  $$\langle\langle sys \rangle\rangle \mathsf{R}^{r}_{\bowtie \in \{max=?, min=?\}}[\mathsf{F}^{\star}\ \phi]$$

  In the context of our game specifications, the above pattern enables the quantification of the maximum/minimum accrued reward r along paths leading to states satisfying ϕ that can be guaranteed by the system player, independently of the strategy followed by the environment. Examples of such properties are $\langle\langle sys \rangle\rangle \mathsf{R}^{\mathsf{utility}}_{max=?}[\mathsf{F}^{\mathsf{c}}\ \mathsf{empty\_batt}]$ ("value of the maximum accrued utility reward that the system can guarantee before the full depletion of the battery, regardless of the strategy followed by the environment"), or $\langle\langle sys \rangle\rangle \mathsf{R}^{\mathsf{time}}_{min=?}[\mathsf{F}^{\mathsf{c}}\ \mathsf{rt} < \mathsf{MAX\_RT}]$ ("value of the minimum time that the system can guarantee to reach an acceptable performance level in which response time rt is below a threshold MAX_RT, regardless of the strategy followed by the environment").

In the next section, we illustrate our approach by describing how the checking of rPATL properties on SMG models can support the analysis of self-adaptive behavior in different contexts.

4 Applications

In this section, we first illustrate how to instance our approach in the context of self-protecting systems, comparing how different variants of system defense mechanisms perform in an environment including attackers. Second, we describe an application of SMG analysis to systematically reason about the involvement of humans in the execution of adaptations in the context of an industrial middleware used to monitor energy production plants.

Our formal models for analyzing self-protecting systems and human-in-the-loop adaptation are implemented in PRISM-games [10], an extension of the probabilistic model-checker PRISM [32] for modeling and analyzing SMGs.

4.1 Self-Protecting Systems

Probabilistic model checking of SMGs can be a powerful tool applied in the context of self-protecting systems [39]. In this context, the interplay between a defending system and a potentially hostile environment including attackers can be modeled as competing players in a SMG. In this section, we illustrate how model checking of SMGs can be used to model variants of self-protecting systems to compare their effectiveness. Concretely, we compare a generic version of Moving Target Defense (MTD) [36] with the use of self-adapting software architectures [40].

The fundamental premise behind MTD is to create a dynamic and shifting system, which is more difficult to attack than a static system. To be considered an MTD system, a system needs to shift the different vectors along which it could potentially be attacked. Such a collection of vectors is often termed an attack surface, and changing the surface in different ways as the system runs makes an attack more difficult because the surface is not fixed. The main motivation behind an MTD system is to significantly increase the cost to adversaries attacking it, while avoiding creating a higher cost to the defender.

In contrast, self-adapting software architectures tackle in a different way the self-protection of software systems by applying specific strategies to increase the complexity of the system, decrease the potential attack surface, or aid in the detection of attacks [37, 40]. In terms of MTD, self-adaptive systems apply approaches at the architectural or enterprise level, meaning that security can be reasoned about in the context of other qualities and broader business concerns.

The results at the end of this section demonstrate the impact of shifting self-protecting mechanisms from working in response to existing stimulus (reactive) to acting in preparation for potential perceived threats based on predictions about the environment (proactive). We consider in our study three variants of self-protection:

– *Uninformed-Proactive*. The defending system adapts proactively with a fixed time frequency based on an internal model of the environment (i.e., it does not factor

in sensed information about the environment in decision making regarding when or which tactics should be performed).
- *Predictive-Proactive*. The system adapts proactively, but factoring in information sensed from the environment, as well as predictions about the environment's future behavior (e.g., trend analysis, or seasonal information).
- *Reactive*. The defending system adapts reactively, executing tactics based on information sensed from the environment (e.g., after a number of detected probing events that can increase the amount of information that a potential attacker might have available, thereby increasing its chances of carrying out a successful attack).

Game description Our formal model for analyzing self-protecting systems is played in turns by two players that are in control of the behavior of the environment (including an attacker), and the defending system, respectively. In this game, the environment's goal is compromising the defending system by carrying out an attack on it. The probability of success of the attack is directly proportional to the amount of information that the attacker has successfully gathered about the system through subsequent probing attempts during the game. On the other hand, the goal of the defending system is thwarting the attacks by adapting the system. The behavior of the system includes a single, abstract adaptation tactic that has the effect of invalidating the information that the attacker has collected about the system up to the point in which the system adapts. Probing, attacking, and adapting are actions that incur costs in terms of consumed resources, both on the attacker's and the defending system's sides.

- *System Player*. The behavior of the defending system is parameterized by the constants shown in Table 2 (note that MAX_THREAT_LEVEL and THREAT_SENSITIVITY are relevant only to the reactive game variant). During its turn, the system can:
  - Yield the turn to the environment player without executing any actions.
  - Adapt, resulting in the (partial) invalidation of the information collected by the attacker. The adaptation of the system can only be executed if the following conditions are satisfied:
    * There must be enough available system resources to carry out the adaptation (ADAPTATION_COST).
    * The perceived threat level must be above the threshold (THREAT_SENSITIVITY). This condition applies only in the reactive variant of the model (the value of the threshold is always set to zero in proactive variants).
    * The system should be able to adapt at the current time. This applies only for the uninformed proactive variant of the model, in which the system is allowed to adapt only with a fixed frequency (ADAPT_PERIOD).

  Once the adaptation commands executes, it carries out a reduction in the amount of information collected by the attacker directly proportional to the value set in the parameter ADAPTATION_EFFECTIVENESS. In the reactive version of the system, the level of perceived threat is also reduced in the same proportion.
- *Environment Player*. The behavior is parameterized by the constants shown in Table 3. During its turn, the environment player can either:

Name	Game variant	Description
MAX_SYSTEM_RES	All	Maximum amount of available system resources.
ADAPTATION_COST	All	Amount of resources consumed each time the system adapts.
ADAPTATION_EFFECTIVENESS	All	Probability of invalidating the information that the attacker ha gathered about the system.
MAX_THREAT_LEVEL	Reactive	Maximum level of threat as perceived by the defending system.
THREAT_SENSITIVITY	Reactive	Degree of reactiveness of the defending system regarding threat detection (e.g., it is the minimum threshold in perceived threat level required to adapt, where threat level is increased by external events such as attacks or probes).

Table 2. Constants parameterizing the behavior of the system player.

- Probe the system if there are enough resources available for it. There are two probabilistic outcomes for the probing action: (i) the probe succeeds with probability P_PROBE_SUCCESS, incrementing the amount of information available to the attacker, as well as the threat level perceived by the system, or (ii) the probe fails with probability 1-P_PROBE_SUCCESS, incrementing only the threat level perceived by the system.
- Attack the system if there are enough resources available for it. The possible outcomes of the attack are: (i) the attack succeeds with probability P_ATTACK_SUCCESS, and (ii) the attack fails with probability 1-P_ATTACK_SUCCESS, raising the value of the threat level perceived by the system.
- Yield the turn to the system player without executing any actions.

Name	Description
MAX_ATTACKER_RES	Maximum amount of resources available to the attacker.
PROBE_COST	Amount of resources consumed when probing the system.
ATTACK_COST	Amount of resources consumed when attacking the system.
PROBE_THREAT_DELTA	Increment in perceived threat level caused by a probe on the system.
ATTACK_THREAT_DELTA	Increment in perceived threat level caused by an attack on the system.
PROBE_INFO_GAIN	Amount of information obtained from successfully probing the system.
P_PROBE_SUCCESS	Probability that a probe on the system will successfully obtain useful information for the attacker.

Table 3. Constants parameterizing the behavior of the environment player.

Analysis Results. To compare the different variants of self-protecting mechanisms, we carried out a set of experiments in which we checked the minimum probability of compromising the system that each of the defense variants could guarantee, independently of the strategy followed by the environment/attacker. This corresponds to quantifying the rPATL property:

$$P_{Comp} \triangleq \langle\langle sys \rangle\rangle \mathsf{P}_{\mathsf{min}=?}[\mathsf{F}\ \mathsf{compromised}]$$

where compromised is a predicate that encodes the occurrence of a successful attack event in the game.

We instanced all the variants of the game model with the set of parameters values displayed in Table 4.1, exploring how P_{Comp} evolved throughout the range of values $[5, 50]$ for available system resources, with the rest of the parameter values fixed.

System		Environment/Attacker	
MAX_SYSTEM_RES	[5,50]	MAX_ATTACKER_RES	5
ADAPTATION_COST	1	PROBE_COST	0
ADAPTATION_EFFECTIVENESS	1	ATTACK_COST	1
MAX_THREAT_LEVEL	5	ATTACK_THREAT_DELTA	2
		PROBE_THREAT_DELTA	1
		PROBE_INFO_GAIN	1
		P_PROBE_SUCCESS	0.8

Table 4. General parameter values for model instantiation.

Specifically, we carried out two experiments:

- *Comparison of uninformed* vs. *predictive variants of proactive adaptation.* Figure 2 shows a comparison of the maximum probability that the attacker has of compromising the system for the two variants that implement proactive defense. The uninformed variant adapts with the maximum possible frequency allowed by the available amount of resources to the system (e.g., if the amount of available resources is 5, and the time frame defined for the game is 50, the system will adapt every 10 time units. The results show that given the same amount of system resources, the predictive variant always performs better than uninformed adaptation.[7] Moreover, while the predictive variant progressively and smoothly reduces the probability of the attacker compromising the system, the uninformed one is more uneven, presenting different intervals during which the addition of system resources does not make any difference concerning the probability of the system being compromised (e.g., the probability does not change for the uninformed variant during the interval $[25, 49]$).
- *Comparison of predictive proactive adaptation* vs. *reactive adaptation*. Figure 3 shows P_{Comp} for the predictive variant of MTD, comparing it with reactive adaptation variants that have different threat sensitivity levels. As expected, predictive proactive adaptation always performs better than the different reactive variants, since the solution space of reactive adaptation is always a subset of the solutions available to predictive proactive adaptation. In particular, it can be observed how increasing levels of sensitivity (i.e., lower values of threshold THREAT_SENSITIVITY) yield increasingly better results.

[7] Our experiments assume that the predictive variant has access to a perfect prediction of the future evolution of the environment.

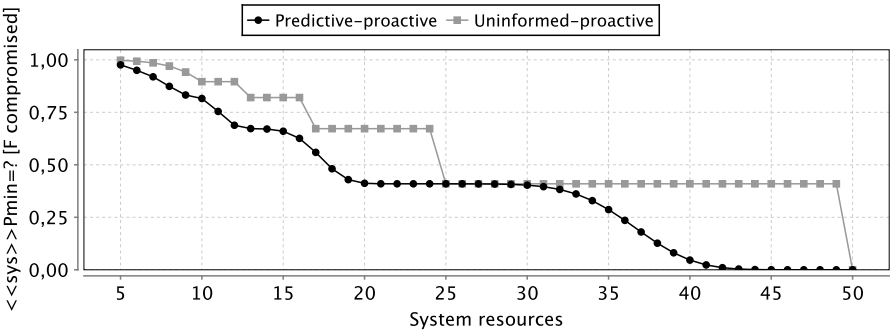

Fig. 2. Probability of compromising the system in proactive adaptation: uninformed *vs.* predictive.

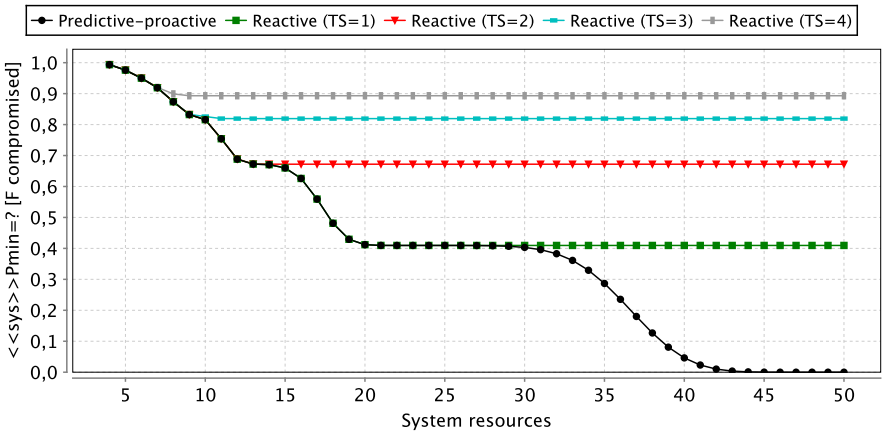

Fig. 3. Probability of compromising the system in proactive *vs.* reactive adaptation.

4.2 Latency-aware Proactive Adaptation

When planning how to adapt, self-adaptive approaches tend to make simplifying assumptions about the properties of adaptation, such as ignoring the time it takes for an adaptation tactic to cause its intended effect. Different adaptation tactics take different amounts of time until their effects are produced. For example, consider two tactics to deal with an increase in the load of a system: reducing the fidelity of the results (e.g., less resolution, fewer elements, etc.), and adding a computer to share the load. Adapting the system to produce results with less fidelity may be achieved quickly if it can be done by changing a simple setting in a component, whereas powering up an additional computer to share the load may take some time. We refer to the time it takes since a tactic is started until its effect is observed as *tactic latency*. Current approaches to decide how to self-adapt do not take the latency of adaptation tactics into account when deciding what tactic to enact. For proactive adaptation, considering tactic latency is necessary so that the adaptation can be started with the sufficient lead time to be ready in time.

In this section, we show how SMG analysis can help to mitigate the uncertainty derived from simplifying assumptions by enabling the comparison of adaptation alternatives that consider tactic latency information for proactive adaptation with those that are latency-agnostic.

Game Description. We model our game for analysis of latency-aware adaptation based on Znn.com [14], a case study portraying a representative scenario for the application of self-adaptation in software systems which has been extensively used to assess different research advances in self-adaptive systems. Znn.com embodies the typical infrastructure for a news website, and has a three-tier architecture consisting of a set of servers that provide contents from backend databases to clients via front-end presentation logic (Figure 4). The system uses a load balancer to balance requests across a pool of replicated servers, the size of which can be adjusted according to service demand. A set of clients makes stateless requests, and the servers deliver the requested contents.

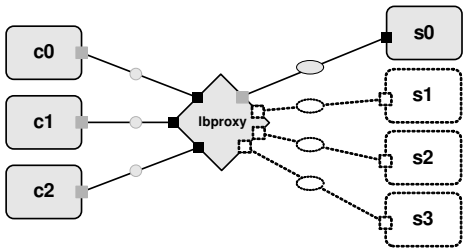

Fig. 4. Znn.com system architecture

The main objective for Znn.com is to provide content to customers within a reasonable response time, while keeping the cost of the server pool within a certain operating budget. It is considered that from time to time, due to highly popular events, Znn.com experiences spikes in requests that it cannot serve adequately, even at maximum pool size. To prevent losing customers, the system can maintain functionality at a reduced level of fidelity by setting servers to return only textual content during such peak times, instead of not providing service to some of its customers. Concretely, there are two main quality objectives for the self-adaptation of the system: (i) performance, which depends on request response time, server load, and network bandwidth, and (ii) cost, which is associated with the number of active servers.

In Znn.com, when response time becomes too high, the system is able to increment its server pool size if it is within budget to improve performance; or switch servers to textual mode if the cost is near to budget limit.[8]

The game is played in turns by two players that are in control of the behavior of the environment and the system, respectively. We assume that the frequency with which the environment updates the value of non-controllable variables, and the system responds

[8] We consider a simple version of Znn.com that adapts only by adjusting server pool size.

to these changes is defined by the constant TAU.[9] Hence, two consecutive turns of the same player are separated by a time period of duration TAU.

- *Environment Player*. The environment is in control of the evolution of time and other variables of the execution context that are out of the system's control (e.g., service requests arriving at the system). During its turn, the environment sets the amount of request arrivals for the current time period and updates the values of other environment variables (e.g., increasing the variable that keeps track of execution time).
- *System Player*. During its turn, the system can trigger the activation of a new server, which will become effective only after the latency period of the tactic expires (modeled using a counter that keeps track of time since the tactic was triggered). Alternatively, the system can discharge a server (with no latency associated). Concretely, the system can execute one of the following actions during its turn:
  - Triggering the activation of a server. This action executes only if the current number of active servers has not reached the maximum allowed, and the counter that controls tactic latency is inactive (meaning that there is not currently a server already booting in the system). Triggering server activation sets the value of the counter to the latency value for the tactic.
  - Effective server activation, which executes only when the counter that controls tactic latency reaches zero, incrementing the number of servers in the system, and deactivating the counter.
  - Deactivation of a server, which decrements the number of active servers. This action is executed only if the current number of active servers is greater than the minimum allowed and the counter for server activation is not active.
  - Yield the turn to the environment player without executing any actions (decreasing the value of the latency counter if it is active).

  In addition, the system updates during its turn the value of the response time according to the request arrivals placed by the environment player during the current time period and the number of active servers (computed using an M/M/c queuing model [16]).

The objective of the system player in the game is maximizing the accrued utility during execution. To represent utility, we employ a reward structure that maps game states to a single instantaneous utility value computed according to a set of utility functions and preferences (i.e., weights). Specifically, we consider two functions that map the response time and the number of active servers in the system to performance and cost utility, respectively.

In latency-aware adaptation, the reward structure rIU encoded in the game employs the real value of response time and number of servers during the tactic latency period to compute the value of instantaneous utility. However, in non-latency-aware adaptation, the instantaneous utility expected by the algorithm during the latency period for activating a server does not match the real utility extracted for the system, since the new server has not yet impacted the performance. In this case, we add to the model a new

[9] TAU is the period of the self-adaptation control loop that includes the monitoring of the environment, and the adaptation decision.

reward structure rEIU in which the utility for performance during the latency period is based on the response time that the system would have if the new server had completed its activation.

Analysis Results. To compare latency-aware *vs.* non-latency-aware adaptation, we make use of rPATL specifications that enable us to analyze (i) the maximum accrued utility that adaptation can guarantee, independently of the behavior of the environment (worst-case scenario).

- *Latency-aware adaptation.* We define the *real guaranteed accrued utility* (U_{rga}) as the maximum real instantaneous utility reward accumulated throughout execution that the system player is able to guarantee, independently of the behavior of the environment player:

$$U_{rga} \triangleq \langle\langle \mathsf{sys} \rangle\rangle \mathsf{R}^{\mathsf{rIU}}_{\mathsf{max}=?}[\mathsf{F}^{\mathsf{c}}\ \mathsf{t} = \mathsf{MAX\_TIME}]$$

  This enables us to obtain the utility that an optimal self-adaptation algorithm would be able to extract from the system, given the most adverse possible conditions of the environment.
- *Non-latency-aware Adaptation.* In non-latency-aware adaptation, real utility does not coincide with the expected utility that an arbitrary algorithm would employ for decision-making, so analysis is carried out in two steps:
  1. Compute the strategy that the adaptation algorithm would follow based on the information it employs about expected utility. That strategy is computed based on an rPATL specification that obtains the expected guaranteed accrued utility (U_{ega}) for the system player:

$$U_{ega} \triangleq \langle\langle \mathsf{sys} \rangle\rangle \mathsf{R}^{\mathsf{rEIU}}_{\mathsf{max}=?}[\mathsf{F}^{\mathsf{c}}\ \mathsf{t} = \mathsf{MAX\_TIME}]$$

     For the specification of this property we employ the expected utility reward rEIU instead of the real utility reward rIU. Note that for latency-aware adaptation $U_{ega} = U_{rga}$.
  2. Verify the U_{rga} under the generated strategy. We do this by building a product of the existing game model and the strategy synthesized in the previous step, obtaining a new game under which further properties can be verified. In our case, once we have computed a strategy for the system player to maximize expected utility, we quantify the reward for real utility in the new game in which the system player strategy has already been fixed.

Table 5 compares the results for the utility extracted from the system by a latency-aware *vs.* a non-latency-aware version of the system player, for two different models of Znn.com that represent an execution of the system during 100 and 200s, respectively. The models consider a pool of up to 4 servers, out of which 2 are initially active. The period duration TAU is set to 10s, and for each version of the model, we compute the results for three variants with different latencies for the activation of servers of up to 3*TAU s. The maximum number of arrivals that the environment can place per time period is 20, whereas the time it takes the system to service every request is 1s.

We define the relative difference between the expected and the real guaranteed utility as:

$$\Delta U_{er} = (1 - \frac{U_{ega}}{U_{rga}}) \times 100$$

Moreover, we define the relative difference in real guaranteed utility between latency-aware an non-latency aware adaptation as:

$$\Delta U_{rga} = (1 - \frac{U^n_{rga}}{U^l_{rga}}) \times 100,$$

where U^n_{rga} and U^l_{rga} designate the real guaranteed accrued utility for non-latency-aware and latency-aware adaptation, respectively.

Table 5. SMG model checking results for Znn

MAX_TIME (s)	Latency (s)	Latency-Aware			Non-Latency-Aware			ΔU_{rga} (%)
		U_{ega}	U_{rga}	ΔU_{er}(%)	U_{ega}	U_{rga}	ΔU_{er}(%)	
100	TAU	53.77	53.77	0	65.97	48.12	-27.05	10.5
	2*TAU	49.35	49.35	0	64.3	42.1	-34.5	14.69
	3* TAU	45.6	45.6	0	64.3	33.25	-48.2	27
200	TAU	110.02	110.02	0	127.25	95.9	-24.63	12.83
	2*TAU	105.6	105.6	0	125.57	76.6	-38.99	27.46
	3* TAU	101.17	101.17	0	123.9	66.15	-46.6	34.61

The results show that latency-aware adaptation outperforms in all cases its non-latency-aware counterpart. Concretely, latency-aware adaptation is able to guarantee an increment in utility extracted from the system, independently of the behavior of the environment (ΔU_{rga}) that ranges between approximately 10 and 34%, increasing progressively with higher tactic latencies. Regarding the delta between expected and real utility that adaptation can guarantee, we can observe that ΔU_{er} is always zero in the case of latency-aware adaptation, since expected and real utilities always have the same value, whereas in the case of non-latency-aware adaptation there is a remarkable decrement that ranges between 24 and 48%, also progressively increasing with higher tactic latency.

4.3 Human-in-the-loop Adaptation

The different activities of the MAPE-K loop in some classes of systems (e.g., safety-critical) and application domains can benefit from human involvement by: (i) receiving information difficult to automatically monitor or analyze from humans acting as sophisticated sensors (e.g., indicating whether there is an ongoing anomalous situation), (ii) incorporating human input into the decision-making process to provide better insight about the best way of adapting the system, or (iii) employing humans as system-level effectors to execute adaptations (e.g., in cases in which full automation is not possible, or as a fallback mechanism).

However, the behavior of human participants is typically influenced by factors external to the system (e.g., training level, stress, fatigue) that determine their likelihood of succeeding at carrying out a particular task, how long it will take, or even if they are willing to perform it in the first place. Without consideration of these factors, it is difficult to decide when to involve humans in adaptation, and in which way.

Answering these questions demands new approaches to systematically reason about how the behavior of human participants, incorporated as integral system elements, af-

fects the outcome of adaptation. In this section, we illustrate how the explicit modeling of human factors in stochastic multiplayer games (SMGs) can be used to analyze human-system-environment interactions to provide a better insight into the trade-offs of involving humans in adaptation. In particular, we focus on the role of human participants as actors (i.e., effectors) during the execution stage of adaptation, and illustrate the approach in the context of DCAS (Data Acquisition and Control Service) [7], a middleware from Critical Software that provides a reusable infrastructure to manage the monitoring of highly populated networks of devices equipped with sensors.

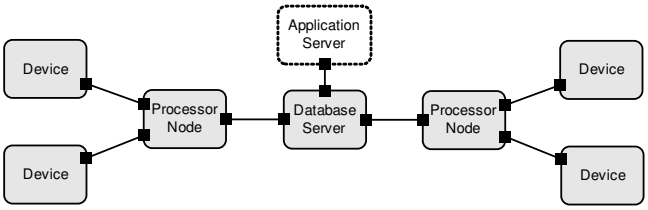

Fig. 5. Architecture of a DCAS-based system.

The basic building blocks in a DCAS-based system (Figure 5) are: [10]

- *Devices* are equipped with one or more sensors to obtain data from the application domain (e.g., from wind towers, or solar panels). Each sensor has an associated *data stream* from which data can be read. Each type of device has its particular characteristics (e.g., data polling rate, or expected value ranges) specified in a *device profile*.
- *Processor nodes* pull data from the devices at a rate configured in the device profile, and dispatch this data to the database server. Each processor node includes a set of processes called *Data Requester Processor Pollers* (DRPPs or *pollers*, for short) responsible for retrieving data from the devices. Communication between DRPPs and devices is synchronous, so DRPPs remain blocked until devices respond to data requests or a timeout expires. This is the main performance bottleneck of DCAS.
- *Database server* stores the data collected from devices by processor nodes.
- *Application server* is connected to the database server to obtain data, which can be presented to the operators of the system or processed automatically by application software. However, DCAS is application-agnostic, so the application server will not be discussed in the remainder of this paper.

The main objective of DCAS is to collect data from the connected devices at a rate as close as possible to the one configured in their device profiles, while making an efficient use of the computational resources in the processor nodes. Specifically, the primary concern in DCAS is providing service while maintaining acceptable levels of performance, measured in terms of processed data requests per second (rps) inserted in the database, while the secondary concern is optimizing the cost of operating the

[10] We herein consider a simplified version of the DCAS architecture. Further details about DCAS can be found in [7].

system, which is mapped to the number of active processor nodes (i.e., each active processor node has a fixed operation cost per time unit).

In situations in which new devices are connected to the network at runtime and all available resources in the set of active processor nodes are already being used, DCAS includes a scale out mechanism aimed at maintaining an acceptable performance level by dynamically activating new processor nodes, according to the demand determined by the new system's workload and operating conditions. Scale out can be performed in two different ways:

- As a *manual process* carried out by a human operator. This is a slow and demanding process, in which a new processor node must be manually deployed, and devices re-attached across the different already active processor nodes, according to the particular situation.
- As an *automated process* that can be executed by adaptation logic residing in an closed control loop. Although this process is faster than the manual version of scale out and does not require any human intervention, it is less effective in terms of exploiting system resources, since only new devices can be attached to the new (predeployed) processor nodes being activated (i.e., devices already attached to other processor nodes cannot be re-attached, restricting the space of target configurations that the system can adopt with respect to the manual variant of scale out).

Game Description. The game is played in turns by two players that are in control of the behavior of the environment and a DCAS-based system, respectively. Concretely, the system player controls of all the actions that belong to a human actor and the target system, including the execution of adaptation tactics for manual and automatic scale out. The objective of the system player is maximizing accrued utility obtained from performance and cost during execution.

- *Environment Player.* Controls the evolution of variables in the execution context that are out of the system's control. For the sake of simplicity, we assume in this case a neutral behavior of the environment that only keeps track of time, although additional behavior controlling other elements (e.g., network delay) can be encoded (please refer to Sections 4.2 and 4.1 for further details illustrating the modeling of adversarial environment behavior).
- *System Player.* Models the cooperative behavior of the system and the human operator. It consists of two parts:
  - *Human model.* Attributes of human actors that might affect interactions with the system are captured in a model inspired by an opportunity-willingness-capability (OWC) ontology described in the context of cyber-human systems [23]. Although a single actor is modeled in our game, system descriptions can incorporate multiple human actor types (e.g., human actor roles specialized in different tasks), each of which can have multiple instances (e.g, operators with different levels of training in a particular task). Attributes of human actor types can be categorized into:
    - *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?).

Example 1. We consider a tactic to have a human actor manually deploy a processor node (addPN) when performing scale out in DCAS. Opportunity elements are $OE^{\mathsf{addPN}} = \{L, B\}$, where L represents the operator's location, and B tells us whether the operator is busy doing something else:

- $L.state \in \{$*operator on location (ONL), operator off location (OFFL)*$\}$.
- $B.state \in \{$*operator busy (OB), operator not busy (ONB)*$\}$.

Using OE^{addPN}, we can define an opportunity function for the tactic $f_O^{\mathsf{addPN}} = (L.state == ONL) \cdot (B.state == ONB)$ that can be used to constrain its applicability only to situations in which there is an operator on location who is not busy.

* *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, willingness elements in the case of the addPN tactic can be defined as $WE^{\mathsf{addPN}} = \{S\}$, where $S.state \in [0, 10]$ represents the operator's stress level. A willingness function mapping willingness elements to a probability of tactic completion can be defined as $f_W^{\mathsf{addPN}} = pr_W(S.state)$, with $pr_W : S \to [0, 1]$.
* 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. In our example, we define capability elements as $CE^{\mathsf{addPN}} = \{T\}$, where T represents the operator's level of training (e.g., $T.state \in [0, 1]$). We define a capability function that maps training level to a probability of successful tactic performance as $f_C^{\mathsf{addPN}} = pr_C(T.state)$, with $pr_C : T \to [0, 1]$.

- *Target System Behavior.* Models the execution of the different tactics on the system. During its turn, the system player can execute two actions per tactic available. We focus on tactic addPN to illustrate how tactic execution is modeled:
  * Tactic trigger happens when: (i) an operator is on location and not busy, (ii) the number of active processor nodes is lower than the maximum available, and (iii) the latency counter for the tactic is zero. As a consequence, the operator is flagged as busy and the latency counter is activated.
  * Tactic completion. When the tactic's latency counter expires, the command can either: (i) update variables for performance and active number of processor nodes according to a successful activation of a processor node with probability addPN_wc_prob (determined by the willingness and capability elements defined in the human model), or (ii) fail to activate the node with probability 1-addPN_wc_prob. In both cases, the latency counter is reset, and the busy status of the operator is set to false.

In addition, the system player can choose not to execute any actions, just updating any tactic active latency counters that have not reached their respective tactic latency values. Note that the encoding used for the automatic scale out tactic (activatePN) follows the same structure, but without any OWC elements encoded in the guards or updates of the commands (activation of a processor node using this tactic is assumed to be always successful).

- *Adaptation Logic.* The adaptation logic placed in the controller is modeled as two alternative adaptation strategies [11] for scale out (scaleOutOp and scaleOut). Actions in the specification of the strategies synchronize with the trigger actions on the specification of the target system behavior on shared action names.
  - scaleoutOp models the variant of the scale out mechanism that makes use of a human actor by first triggering tactic addPN. Once an observation period for the effects of the tactic has expired, the strategy falls back on automatic activation by triggering the activatePN tactic if the activation of the new processor node by the human operator has not been successful.
  - scaleOut models the automatic scale out mechanism with a single command that triggers the execution of tactic activatePN on the target system.

The game includes an encoding of utility functions and preferences for performance and cost as a reward structure rGU that enables the quantification of instantaneous utility in game states.

Analysis Results. We exploit our human-in-the-loop adaptation game model to deal with the uncertainty derived from involving humans in adaptation, employing its analysis to determine: (i) the expected outcome of human involvement in adaptation, and (ii) the conditions under which such involvement improves over fully automated adaptation.

– *Strategy Utility.* The expected utility value of an adaptation strategy (potentially including non-automated tactics) is quantified by checking the reachability reward property:

$$u_{mau} \triangleq \langle\langle \mathsf{sys} \rangle\rangle \mathsf{R}^{\mathsf{rGU}}_{\mathsf{max}=?}[\mathsf{F}^{\mathsf{c}}\ \mathsf{t}{=}\mathsf{MAX\_TIME}]$$

The property obtains the maximum accrued utility value (i.e., corresponding to reward rGU) that the system player can achieve until the end of execution (t=MAX_TIME). Figure 6(a) depicts strategy utility analysis results for the different adaptation strategies in a scale out DCAS scenario. In the figure, a discretized region of the state space is projected over the dimensions that correspond to training and stress levels of a human actor (with values in the range [0,1] and [0,10], respectively). Each point on the mesh represents the maximum accrued utility that the system can achieve on a DCAS SMG model instanced for a time frame [0,200].
If we focus on the curve described by values in the lowest stress level, the figure shows how the use of strategy scaleOutOp attains values that go above 140 for maximum training, whereas values below 0.4 are highly penalized and barely yield utility. Moreover, progressively higher stress levels reduce the probability of successful tactic completion, flattening the curve to a point in which training does not make any difference and the strategy yields no utility. On the contrary, the utility obtained by the automated scaleOut strategy (represented by the plane in the figure) is always constant since it is not affected by willingness or capability factors.

[11] In this context we refer to adaptation strategies as described in Stitch [15]. These correspond to decision trees in which branches are defined by means of condition-action-delay rules.

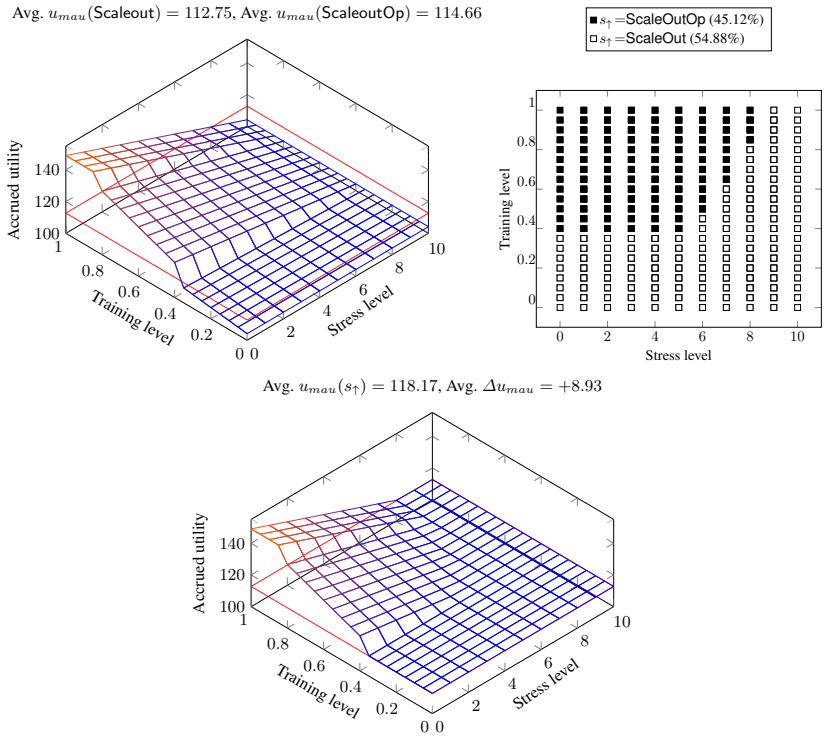

Fig. 6. Experimental results: (a) ScaleOut *vs* ScaleOutOp strategy utility (top left), (b) strategy selection (top right), and (c) combined utility (bottom).

– *Strategy Selection.* Given a repertoire of adaptation strategies $\mathcal{S}$, we can analyze their expected outcome in a given situation by computing their expected accrued utility according to the procedure described above. Based on this information, the different strategies can be ranked to select the one that maximizes the expected outcome in terms of utility. Hence the selected strategy $s_\uparrow$ can be determined according to:

$$s_\uparrow \triangleq \arg\max_{s \in \mathcal{S}} u_{mau}(s)$$

where $u_{mau}(s)$ is the value of property u_{mau} evaluated in a model instantiated with the adaptation logic of strategy s.

Figure 6(b) shows the results of the analysis of strategy selection in DCAS scale out. The states in which human involvement via strategy scaleOutOp is chosen ($\simeq 45\%$ of states) are represented in black, whereas states in which the fully automated strategy scaleOut is selected ($\simeq 55\%$) are colored in white. The figure shows that human involvement is only advisable in areas in which the operator has a stress level of 8 and below. Progressively higher stress levels make human involvement preferable only when also progressively higher training levels exist, which is consistent with maximizing the probability of successful adaptation tactic completion. In any case, for training levels below 0.4, human actor participation is not selected even with zero

stress level (this is consistent with the function f_C^{addPN} that we define for the capability elements of the human actor, which highly penalizes poorly trained operators). Figure 6(c) shows the combined accrued utility mesh that results from the selection process (i.e, every point in the mesh is computed as $u_{mau}(s_{\uparrow})$). Note that the minimum accrued utility never goes below the achievable utility level of the automatic approach, over which improvements are made in the areas in which the strategy involving human actors is selected. The average improvement in the combined solution corresponds to a percentual improvement of 7.94% over the automatic scale out approach, and 7.81% over the manual one.

5 Conclusions and Future Work

In this chapter, we have described an approach that enables developers to approximate the behavioral envelope of a self-adaptive system by analyzing best- and worst-case scenarios of alternative designs for self-adaptation mechanisms. The approach relies on probabilistic model checking of stochastic multiplayer games (SMGs), and exploits specifications that encode assumptions about the behavior of the environment, which are used as constraints for the generated state-space.

The approach can accommodate different levels of detail in the specification of the environment. On the one hand, rich specifications of environment assumptions can help to reduce the game's state-space and provide a better approximation of the system's behavioral envelope. On the other hand, an under-specified set of assumptions will result in a larger game state-space due to an over-approximation of the environment's behavior. However, one of the benefits of the approach is that the guarantees given with respect to the satisfaction of the system's objectives can still be relied upon with an under-specified set of assumptions in worst-case scenario analysis. This stems from the fact that the most adverse environment strategy in the over-approximation will always represent a lower bound in the system's guaranteed payoff (either in terms of probability or reward) with respect to any strategies that can be synthesized for more constrained versions of the environment's behavior.

A second advantage of our approach is that it is purely declarative, enabling self-adaptive system developers to obtain a preliminary understanding of adaptation behavior without the need to have any specific adaptation algorithms or infrastructure in place. This represents a reduced upfront investment in terms of effort when compared to other sources of evidence, such as simulation or prototyping.

We have illustrated the versatility of the approach by showing how it can be used to deal with the uncertainty associated with different sources in various contexts, such as self-protecting systems (parameters over time), proactive latency-aware adaptation (simplifying assumptions), and human-in-the-loop adaptation.

A current limitation of the approach is that its scalability is limited by PRISM-games, which currently uses explicit-state data structures and is to the best of our knowledge the only tool supporting model-checking of SMGs. This limitation can be mitigated in some cases by carefully choosing the level of abstraction and relevant aspects of the system and environment in the model. Moreover, we expect that the maturation of this technology will result in the development of symbolic SMG model checkers that will improve scalability.

Regarding future work, we plan to instantiate our adaptation analysis technique in other contexts to deal with uncertainty in cyber-physical and decentralized systems. As regards the application of the technique to human-in-the-loop adaptation, our current models assume that actors and system are working in coalition to achieve goals. In fact, the interaction may be more subtle than that; Eskins and Sanders point out that humans may have their own motivations that run counter to policy [23]. To capture this subtlety, we plan on extending the encoding of SMGs to model human actors as separate players. Moreover, we will extend the human-in-the-loop instance of the approach to formally model and analyze human involvement in other stages of MAPE-K, studying how to best represent human-controlled tactic selection, and human-assisted knowledge acquisition. Concerning latency-aware adaptation, we aim at exploring how tactic latency information can be further exploited to attain better results both in proactive and reactive adaptation (e.g., by parallelizing tactic executions). Finally, we also aim at refining the approach to do runtime synthesis of proactive adaptation strategies.

6 Acknowledgements

This work is supported in part by awards N000141310401 and N000141310171 from the Office of Naval Research, CNS-0834701 from the National Science Foundation, and by the National Security Agency. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Office of Naval Research or the U.S. government. This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. This material has been approved for public release and unlimited distribution. (DM-0002084).

References

1. R. Alur et al. Alternating-time temporal logic. *J. ACM*, 49(5), 2002.
2. 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.
3. A. Bianco and L. de Alfaro. Model checking of probabalistic and nondeterministic systems. In *FSTTCS*, volume 1026 of *LNCS*. Springer, 1995.
4. V. A. Braberman, N. D'Ippolito, N. Piterman, D. Sykes, and S. Uchitel. Controller synthesis: from modelling to enactment. In D. Notkin, B. H. C. Cheng, and K. Pohl, editors, *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pages 1347–1350. IEEE / ACM, 2013.
5. R. Calinescu et al. Dynamic QoS Management and Optimization in Service-Based Systems. *IEEE Trans. Software Eng.*, 37(3), 2011.
6. R. Calinescu and M. Z. Kwiatkowska. Using Quantitative Analysis to Implement Autonomic IT Systems. In *ICSE*, 2009.
7. J. Cámara, P. Correia, R. de Lemos, D. Garlan, P. Gomes, B. R. Schmerl, and R. Ventura. Evolving an adaptive industrial software system to use architecture-based self-adaptation.

In M. Litoiu and J. Mylopoulos, editors, *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS 2013, San Francisco, CA, USA, May 20-21, 2013*, pages 13–22. IEEE / ACM, 2013.
8. J. Cámara, G. A. Moreno, and D. Garlan. Stochastic game analysis and latency awareness for proactive self-adaptation. In G. Engels and N. 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.
9. J. Cámara, G. A. Moreno, and D. Garlan. Reasoning about human participation in self-adaptive systems. In B. Schmerl and P. 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. To appear.
10. T. Chen et al. PRISM-games: A model checker for stochastic multi-player games. In *TACAS*, volume 7795 of *LNCS*. Springer, 2013.
11. T. Chen, V. Forejt, M. Z. Kwiatkowska, D. Parker, and A. Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92, 2013.
12. T. Chen and J. Lu. Probabilistic alternating-time temporal logic and model checking algorithm. In *FSKD*, volume 2, 2007.
13. B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. D. M. Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle. Software engineering for self-adaptive systems: A research roadmap. In B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2009.
14. S. Cheng, D. Garlan, and B. R. Schmerl. Evaluating the effectiveness of the rainbow self-adaptive system. In *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2009, Vancouver, BC, Canada, May 18-19, 2009*, pages 132–141. IEEE, 2009.
15. S.-W. Cheng and D. Garlan. Stitch: A language for architecture-based self-adaptation. *Journal of Systems and Software*, 85(12), 2012.
16. R. Chiulli. *Quantitative Analysis: An Introduction*. Automation and production systems. Taylor & Francis, 1999.
17. A. Elkhodary, N. Esfahani, and S. 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.
18. 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.
19. I. Epifani et al. Model Evolution by Run-Time Parameter Adaptation. In *ICSE*. IEEE CS, 2009.
20. I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model evolution by run-time parameter adaptation. In J. M. Atlee and P. Inverardi, editors, *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings*, pages 111–121. IEEE, 2009.
21. N. Esfahani, E. Kouroshfar, and S. 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.
22. N. Esfahani and S. Malek. Uncertainty in self-adaptive software systems. In R. de Lemos, H. Giese, H. Muller, and M. Shaw, editors, *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *Lecture Notes in Computer Science*, pages 214–238. Springer, 2013.

23. D. Eskins and W. 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.
24. V. Forejt et al. Automated verification techniques for probabilistic systems. In *SFM*, volume 6659 of *LNCS*. Springer, 2011.
25. D. Garlan, S. Cheng, A. Huang, B. R. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *IEEE Computer*, 37(10):46–54, 2004.
26. R. P. Goldman, D. J. Musliner, and K. D. Krebsbach. Managing online self-adaptation in real-time environments. In *Self-Adaptive Software: Applications*, volume 2614 of *LNCS*, pages 6–23. Springer, 2003.
27. W. V. D. Hoek and M. Wooldridge. Model checking cooperation, knowledge, and time - a case study. In *Research in Economics*, 2003.
28. M. C. Huebscher and J. A. McCann. A survey of autonomic computing - degrees, models, and applications. *ACM Comput. Surv.*, 40(3), 2008.
29. J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36, 2003.
30. A. N. Kolmogorov. *Foundations of the Theory of Probability*. Chelsea, New York, 1956.
31. S. Kremer and J.-F. Raskin. A game-based verification of non-repudiation and fair exchange protocols. In *CONCUR 2001*, volume 2154 of *LNCS*. Springer, 2001.
32. M. Kwiatkowska et al. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. of CAV'11*, volume 6806 of *LNCS*. Springer, 2011.
33. M. Kwiatkowska, G. Norman, and D. Parker. Stochastic games for verification of probabilistic timed automata. In J. Ouaknine and F. 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.
34. W. Li, D. Sadigh, S. Sastry, and S. Seshia. Synthesis for human-in-the-loop control systems. In E. Abraham and K. 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.
35. R. B. Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, Cambridge, MA, 1991.
36. H. Okhravi, M. Rabe, T. Mayberry, W. Leonard, T. Hobson, D. Bigelow, and W. Streilein. Survey of cyber moving target techniques. Technical Report 1166, Lincoln Laboratory, Massachusetts Institute of Technology, 2013.
37. B. Schmerl, J. Camara, J. Gennari, D. Garlan, P. Casanova, G. A. Moreno, T. J. Glazier, and J. M. Barnes. Architecture-Based Self-Protection: Composing and Reasoning about Denial-of-Service Mitigations. In *Symposium and Bootcamp on the Science of Security (HotSoS)*, Raleigh, USA, 8-9 April 2014.
38. J. Whittle, P. Sawyer, N. Bencomo, B. 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.
39. E. Yuan, N. Esfahani, and S. Malek. A systematic survey of self-protecting software systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 8(4):17, 2014.
40. E. Yuan, S. Malek, B. Schmerl, D. Garlan, and J. Gennari. Architecture-based self-protecting software systems. In *Proceedings of the Ninth International ACM Sigsoft Conference on the Quality of Software Architectures (QoSA 2013)*, 17-21 June 2013.
41. L. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338 – 353, 1965.
42. L. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 100, Supplement 1(0):9 – 34, 1999.

43. X. Zhang and C. Lung. Improving software performance and reliability with an architecture-based self-adaptive framework. In S. I. Ahamed, D. Bae, S. D. Cha, C. K. Chang, R. Subramanyan, E. Wong, and H. Yang, editors, *Proceedings of the 34th Annual IEEE International Computer Software and Applications Conference, COMPSAC 2010, Seoul, Korea, 19-23 July 2010*, pages 72–81. IEEE Computer Society, 2010.