Evaluating Trade-Offs of Human Involvement in Self-Adaptive Systems

Javier Cámara*, David Garlan*, Gabriel A. Moreno+, and Bradley Schmerl*

* *Institute for Software Research. Carnegie Mellon University. Pittsburgh, PA 15213, USA*

+ *Software Engineering Institute. Carnegie Mellon University. Pittsburgh, PA 15213, USA*

Abstract

Software systems are increasingly called upon to autonomously manage their goals in changing contexts and environments, and under evolving requirements. In some circumstances, autonomous systems cannot be fully-automated but instead cooperate with human operators to maintain and adapt themselves. Furthermore, there are times when a choice should be made between doing a manual or automated repair. Involving operators in self-adaptation should itself be adaptive, and consider aspects such as the training, attention, and ability of operators. Not only do these aspects change from person to person, but they may change with the same person. These aspects make the choice of whether to involve humans non-obvious. Self-adaptive systems should trade-off whether to involve operators, taking these aspects into consideration along with other business qualities it is attempting to achieve. In this chapter, we identify the various roles that operators can perform in cooperating with self-adapting systems. We focus on humans as effectors - doing tasks which are difficult or infeasible to automate. We describe how we modified our self-adaptive framework, Rainbow, to involve operators in this way, which involved choosing suitable human models and integrating them into the existing utility trade-off decision models of Rainbow. We use probabilistic modeling and quantitative verification to analyze the trade-offs of involving humans in adaptation, and complement our study with experiments to show how different business preferences and modalities of human involvement may result in different outcomes.

Keywords: Self-Adaptation, Human-in-the-Loop, Adaptive Autonomy, Trade-Offs

1. Introduction

Modern society increasingly relies upon software-intensive systems to support a wide range of tasks in multiple application domains, such as energy, transportation, and communications. Despite the critical nature of many of these systems, it is increasingly difficult to obtain guarantees about their ability to provide service that can justifiably be trusted in the presence of changes happening in their environment (e.g., resource availability), or within the system itself (e.g., faults). The growing complexity of these systems and the high degree of uncertainty in the environment in which they typically have to operate are two of the main factors that contribute to their lack of predictability.

Early attempts to address this situation involved human oversight, which is expensive and has often been considered as unreliable due to the fact that humans are liable to err and have difficulty to react in a timely manner when situations that demand changes to the system at runtime arise.

In contrast, approaches developed over the last decade in the area of self-adaptive systems [1, 2, 3] advocate for the full automation of mechanisms to adapt the structure and behavior of a system at runtime to overcome some of the limitations associated with human oversight. Self-adaptive approaches often rely on closed-loop control, eliminating the human factor from the solution.

Although fully automated adaptation has proven successful in different application domains, this class of approach may be suboptimal in some situations (e.g., when information required for decision-making is difficult to capture and/or analyze), or may simply be insufficient to effect changes in the system (e.g., when adaptations involve physical changes that cannot be automated).

Among self-adaptive approaches, one of the most successful paradigms to date is MAPE-K, which includes activities to monitor and analyze a software system and its environment, and if the situation demands it, plan and execute adaptations. MAPE-K systems rely on a knowledge base that can include models of a system's environment, goals, and architecture [3, 4]. The different activities in the MAPE-K loop can benefit from human involvement in a variety of ways:

- Monitoring and analysis can receive information from humans (acting as sophis-

ticated sensors) that would be otherwise difficult to automatically monitor or analyze (e.g., humans can indicate whether there is an ongoing anomaly based on context information that is not captured by the models included in the knowledge base).

- Planning can incorporate into the decision-making process input (e.g., recommendations, validation) from application domain experts who can have additional insight about the best way of adapting the system.

- Execution can employ humans as system-level effectors to execute adaptations when changes to the system cannot be fully automated, or as a fallback mechanism.

Despite the benefits that involving humans in adaptation can bring, it is worth noticing that their behavior is influenced by factors external to the system that affect their effectiveness at carrying out different tasks, such as fatigue, or training level. These factors need to be carefully considered if we want to enable systems to discriminate situations in which human involvement is preferable over fully automated adaptations.

Analyzing the trade-offs of involving humans in adaptation demands new approaches to systematically reason about the way in which the behavior of human participants affects the outcome of adaptations. In this chapter, we describe a formal framework to analyze trade-offs in self-adaptation at two different levels: (i) reasoning about business concerns in the context of other (potentially conflicting) business properties; and (ii) reasoning about the effectiveness of automated *vs.* human-driven adaptations with respect to a set of business concerns and preferences.

The core of the framework consists of an extended version of a language to express adaptation models with elements that capture some of the main factors affecting human behavior. Moreover, we show how adaptation models expressed in this language can be encoded as stochastic multiplayer games (SMGs), a formalism amenable to automated verification that can be employed to analyze human-system-environment interactions.

We explore the topics discussed in this chapter using an extension of the Stitch language [5] employed by the Rainbow framework for self-adaptation [4] with el-

ements inspired from opportunity-willingness-capability models employed in *cyber-human systems* [6] that capture major factors that influence human-system interactions. Moreover, we illustrate our approach in the domain of security, employing as motivating scenario the mitigation of denial of service (DoS) attacks in Znn.com, a simple news site system that has been extensively used to assess different research advances in self-adaptive systems.

In the remainder of this chapter, Section 2 describes the example that we employ to illustrate our approach, and Section 3 discusses related work. Section 4 provides an overview of trade-off analysis in self-adaptation as embodied in Stitch. Next, Section 5 describes our human model and its integration in adaptation models described using Stitch. Section 6 describes probabilistic modeling and analysis of adaptation models including humans in the loop. Finally, Section 7 concludes the chapter, indicating research avenues to explore in future work.

2. Motivating Scenario

Before detailing the formal framework to reason about trade-offs of human involvement in adaptation, we introduce an example that will be used to illustrate the approach.

Znn.com [7], is a case study portraying a representative scenario for the application of self-adaptation in software systems embodying the typical infrastructure for a news website. It 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 1). 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.

From time to time, Znn.com can experience spikes in requests that it cannot serve adequately, even at maximum pool size. These spikes can result either from legitimate client traffic caused by a popular event (*slashdot effect*), or by DoS attacks in which malicious clients try to overwhelm system capacity in order to render system services unavailable.

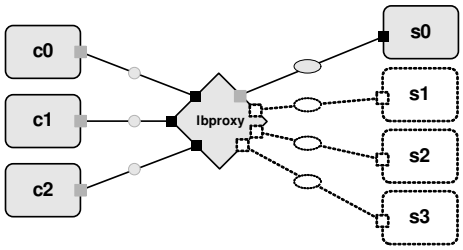

Figure 1: Znn.com system architecture

2.1. System Objectives

Regarding Znn.com's objectives, users of the system are concerned with experiencing service without any disruptions, whereas the organization is interested in minimizing the cost of operating the infrastructure (including not incurring additional operating costs derived from DoS attacks). For users, service disruption can be mapped to specific runtime conditions such as (i) experienced response time for legitimate clients, and (ii) user annoyance, often related to disruptive side effects of defensive tactics. For the organization, we map cost to the specific resources being operated in the infrastructure at runtime (e.g., number of active servers). Moreover, in addition to keeping costs below budget, the organization is also interested in minimizing the fraction of that cost that corresponds to resources consumed by malicious clients. Hence, we identify minimizing the presence of malicious clients as an additional objective.

In short, we identify four quality objectives for Znn.com: (legitimate) client response time (R), user annoyance (A), cost (C), and client maliciousness (M).

2.2. Adaptation Mechanisms

When response time becomes too high due to spikes in requests, the system can employ two general approaches for dealing with the situation: absorb the excess of traffic or suppress it. While the former approach is better suited to situations in which legitimate user traffic has increased due to a popular event, the latter is indicated for dealing with DoS attacks.

Znn.com can absorb the excess traffic employing the tactics: (i) *adding capacity*, which commissions a new replicated web server to share the load; and (ii) *reducing*

service, which reduces the level of service to *text only* (Znn.com has two fidelity levels: *high*, which includes full multimedia content; and *text only*, which does not provide any multimedia content). These tactics are *fully automated*, and are good at improving the performance of the system without compromising the experience of legitimate users. However, employing these tactics comes at a price, since they do not deal with reducing the cost derived from resources consumed by potentially malicious clients, and they can even result in an increment of the cost of operating the system (in the case of adding capacity).

Alternatively, Znn.com can eliminate the excess traffic by enacting the tactics: (i) *blackholing*, which adds the IP addresses of clients that are determined to be attacking the system to a blacklist that blocks their requests; and (ii) *throttling*, which limits the rate of requests accepted from potentially malicious clients. Eliminating excess traffic from potentially malicious clients is an approach that requires accurately identifying the attackers to be effective. Znn.com relies upon the judgement of a human operator to enact these tactics. In general, well-trained operators will be effective at eliminating traffic from malicious clients, but poorly trained operators can increase user annoyance if they cause service disruption to legitimate clients due to mistakes in malicious client identification.

3. Related Work

Deciding whether humans should be involved in the execution of adaptation is no easy task, since their behavior and the outcome of their actions can be affected by transient factors such as changing levels of attention and load, fixed attributes (e.g., level of expertise in carrying out a particular task), or even their physical context (e.g., access to different locations, timing issues). These factors constitute an additional source of uncertainty affecting the self-adaptive system (acknowledged by Esfahani and Malek as *uncertainty due to human in the loop* [8]) that needs to be managed if we want to answer the following questions:

Q1: How can the outcome of adaptation be predicted if human actors are involved in its execution?

Q2: How can it be determined whether human actors should be involved in adaptation?

While answering **Q1** calls for employing models of human characteristics sufficient for representing the probabilistic nature of human behavior and its interaction with the system, **Q2** also demands exploring mechanisms suitable to compare candidate solutions that might include human-system collaborations, as well as fully automated adaptations.

Some existing approaches in self-adaptation that automatically generate adaptation plans at runtime are able to rank candidate solutions by analyzing trade-offs among different qualities [9] or consider uncertainty when tuning the operation of the system (e.g., by dynamically adjusting parameters [10, 11]). However, there are no approaches to the best of our knowledge able to rank candidate solutions by factoring in uncertainty, rendering these approaches insufficient for generating adaptation plans involving humans.

Other approaches in self-adaptation that rely on selection of adaptation strategies defined by a designer at development time [4, 12] are also able to rank candidate solutions by analyzing trade-offs among different quality concerns. Moreover, these approaches are sometimes able to deal with some aspects of uncertainty and timing [4]. These proposals are limited to ranking and selection of fully automated adaptations, since the knowledge models they employ are unable to capture the multiple facets of uncertainty derived from human behavior that affect the outcome of adaptations.

While all the aforementioned approaches focus on fully automated adaptations, Dorn and Taylor [13] introduce a framework that enables a system adaptation manager to reason about the effects of software-level changes on human interactions and vice-versa by mapping a model of what they describe as *human architecture* (described in a language called hADL) to a model of the system's architecture updated at runtime. This approach focuses on the collaboration topology and is able to rank collaboration-(un)aware adaptations to select the best course of action, although it does not explicitly consider uncertainty derived from human behavior as a major factor affecting the outcome of adaptations.

Outside of the scope of self-adaptive systems, some approaches in the business process modeling domain include some aspects of human involvement, providing constructs for describing human activities in business processes and their dependencies [14, 15]. These languages target primarily service-oriented architectures and have limited or no support for other common architectural styles.

Eskins and Sanders [6] introduce a definition of a cyber-human system (CHS) and the opportunity-willingness-capability (OWC) ontology for classifying CHS 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, reflecting the probabilistic nature of human behavior.

If we contrast questions **Q1** and **Q2** with the characteristics of the related approaches described in this section, we can list a set of requirements that a suitable approach to our problem should satisfy:

R1: The approach must include a value system that enables candidate solution ranking, allowing context-sensitive adaptation.

R2: The approach must be able to consider uncertainty as a primary factor that conditions the effectiveness of tasks or adaptations.

R3: The approach must consider timing delays that capture the notion of task or adaptation latency.

R4: The approach must be able to represent and enable the analysis of human participant behavior.

R5: The approach must provide support for a variety of architectural styles.

Although the approaches described in this section partially satisfy these requirements (see Table 1), in this chapter we propose an approach that combines the strengths of the Rainbow framework for self-adaptation [4] and the OWC ontology described in [6]. On the one hand, Rainbow includes a value system based on utility to rank candidate adaptations, explicit time delays to observe the effects of adaptation actions executed on the target system, and it is able to account for uncertainty in the selection

Area	Approach	R1	R2	R3	R4	R5
Self-adaptive Systems	Sykes et al. [9]	✓				✓
	Calinescu et al. [10]		✓	✓		
	Epifani et al. [11]		✓	✓		
	Cheng et al. [4]	✓	✓	✓		✓
	Oreizy et al. [12]	✓				✓
	Dorn & Taylor [13]	✓			✓	✓
Business Process Modelling	BPMN [15]			✓	✓	
	WSBPEL4People [14]			✓	✓	
Cyber-Human Systems	Eskins & Sanders [6]		✓		✓	

Table 1: Requirements satisfied by related approaches.

of adaptive actions. On the other hand, OWC models provide the concepts required to capture human factors that can influence adaptation, some of which are of a probabilistic nature.

In previous work [16], we presented an analysis technique based on model checking of SMGs to quantify the potential benefits of employing different types of algorithms for 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 not latency-aware. In this paper, we apply this analysis technique to the context of human-in-the-loop adaptation, extending SMG models with elements that encode an extended version of Stitch adaptation models with OWC constructs.

4. Analyzing Trade-Offs in Self-Adaptation

In this section, we first introduce the main concepts behind the Stitch language for self-adaptation, showing how elevating the reasoning to an architectural level can provide a principled basis for analyzing the trade-offs among potentially conflicting business objectives. Then, we present a candidate model for quantifying how human involvement in adaptation can affect business objectives. This model is inspired by the OWC ontology described in [6]. Finally, we describe how the concepts behind Stitch and the proposed OWC model can be combined to capture descriptions of adaptations that involve collaborations among the system and human participants.

4.1. Adaptation Model

Although many proposals that rely on closed-loop control exploit architectural models for adaptation [4, 12, 17], in this chapter we use some of the high-level concepts in Stitch [5] as a reference framework to illustrate our approach. Stitch is the language employed by the Rainbow framework [4] to describe automated repairs based on an architectural description of the underlying target system. Rainbow has among its distinct features an explicit architecture model of the target system, a collection of adaptation tactics, and utility preferences to guide adaptation.

We assume a model of adaptation that represents adaptation knowledge employing the following high-level concepts:[1] (i) tactics, or primitive actions that correspond to a single step of adaptation; (ii) strategies, which encapsulate an adaptation process, where each step is the conditional execution of a tactic; and (iii) utility profile, which drives the selection of strategies at runtime based on a set of utility functions and preferences.

4.1.1. Tactic

A tactic is a primitive action that corresponds to a single step of adaptation. Tactics require three parts to be specified: (1) the *condition*, which specifies when a tactic is applicable; (2) the *action*, which defines the script for making changes to the system; and (3) the *effect*, which specifies the expected effect that the tactic will have.

Listing 1 shows an example tactic for activating a set of servers in Znn.com. Line 3 specifies the applicability condition, which says that the tactic may be executed if (i) there is a client experiencing a response above the maximum acceptable threshold (predicate cHiRespTime defined in line 1), and (ii) there are enough servers available to activate. Lines 4-7 specify the action, which is to select a set of servers among those currently inactive (line 5), and enable them (line 6). Line 8 states that the intended effect of the tactic is achieved only if all clients experience a response time below the maximum acceptable threshold.

[1] We use a simplified version of Stitch [5] to illustrate the main ideas in this chapter.

```
1 define boolean cHiRespTime = exists c:T.ClientT in M.components | c.experRespTime>M.MAX_RESPTIME;
2 tactic enlistServers (int n) {
3     condition { cHiRespTime && set.Size(s : T.ServerT in M.components | !s.isArchEnabled)>=n;}
4     action {
5         set servers = Set.randomSubset(Model.findServices(T.ServerT), n);
6         for (T.ServerT freeSvr : servers) { M.enableServer (freeSvr, true); }
7     }
8     effect { !cHiRespTime; }
9 }
```

Listing 1: Tactic for activating a server in Znn.com.

Tactics have an associated cost/benefit impact on the different dimensions of concern in the system. Table 2 shows the impact on different properties of the tactics employed in Znn.com, as well as an indication of how the tactic affects the utility for every particular dimension of concern (the number of upward or downward arrows is directly proportional to the magnitude of utility increments and decrements, respectively).[2] While all tactics reduce the response time experienced by legitimate clients, some of them (e.g., enlistServers and blackholeAttacker) cause a more drastic reduction, resulting in higher utility gains in that particular dimension. Regarding the presence of malicious clients, tactics blackholeAttacker is the most effective, whereas other tactics (*e.g.*, enlistServers) do not have any impact. With respect to cost, strategy enlistServers increases the operating cost and reduces utility in this dimension, since it employs additional resources to absorb incoming traffic. Finally, tactics blackholeAttacker and throttleSuspicious impact negatively on user annoyance, since there is a risk that incorrect detection of malicious clients will lead to annoying a fraction of legitimate clients by blackholing or throttling them.

[2]Note that, to obtain the impact on the different quality dimensions of tactics in practice, the approach relies on expert knowledge or field data about similar existing systems, although nothing prevents the use of machine learning techniques to obtain that information. In this chapter we consider fixed cost/benefit impacts for illustration purposes, although Stitch also supports the specification of sophisticated impact models that are context-sensitive, and can capture probabilistic aspects in the outcome of tactic executions [18].

Tactic	Response Time (R)		Malicious Clients (M)		Cost (C)		User Annoyance (A)	
	Δ Avg. Resp. Time (ms)	ΔU_R	Δ Mal. Clients (%)	ΔU_M	Δ Cost (usd/hr)	ΔU_C	Δ U. Annoyance (%)	ΔU_A
enlistServers	-1000	↑↑↑	0	=	+1.0	↓↓↓	0	=
lowerFidelity	-500	↑↑	0	=	-0.1	↑	0	=
blackholeAttacker	-1000	↑↑↑	-100	↑↑↑	0	=	+50	↓↓
throttleSuspicious	-500	↑↑	0	=	0	=	+25	↓

Table 2: Tactic cost/benefit on qualities and impact on utility dimensions

4.1.2. Strategy

A strategy encapsulates an adaptation process, where each step is the conditional execution of a tactic. Strategies are characterized in Stitch as a tree of condition-action-delay decision nodes, where delays correspond to a time window for observing tactic effects. System feedback (through the dynamically-updated architectural model of the system) is used to determine the next tactic at every step during strategy execution.

```
1 strategy Outgun [cHiRespTime] {
2     t0: (cHiRespTime) –> enlistServers(1) @[30000 /*ms*/] {
3         t1: (success) –> done;
4         t2: (fail) –> lowerFidelity() @[2000 /*ms*/] {
5             t2a: (success) –> done;
6             t2b: (fail) –> TNULL;
7         }
8     }
9 }
```

Listing 2: Strategy for absorbing excess traffic.

Listing 2 shows the Stitch code for a simple adaptation strategy in Znn.com that deals with degraded performance by activating additional servers and reducing the fidelity of the contents served: line 1 specifies the applicability condition that needs to be satisfied for the strategy to be eligible for execution (in this case, predicate cHiRespTime indicates that there are clients experiencing a response time above the acceptable threshold). In the body of the strategy, node t0 (line 2) executes tactic enlistServers if the guard cHiRespTime evaluates to true. To account for the delay in observing the outcome of tactic execution in the system (settling time), t0 specifies a time window of 30 seconds, after which, if the tactic's intended effect (as defined in the tactic script – Listing 1, line 8) is observed, successful tactic completion (keyword success, line 2) leads

to the end strategy execution in normal conditions through node t1 (keyword done). Otherwise, if the intended tactic effect is not observed after the delay window expires (keyword fail, line 4), the strategy attempts to reduce response time by executing the tactic lowerFidelity and waiting 2 seconds to observe its effect, exiting through node t2a if the tactic succeeds. If the intended effect of lowerFidelity is not observed, the strategy exits with an error status via node t2b (line 6).

4.1.3. Utility Profile

In Stitch, the selection of strategies at runtime is driven by utility functions and preferences, which are sensitive to the context of use and able to consider trade-offs among multiple potentially conflicting objectives. The different qualities of concern are characterized as utility functions that map them to architectural properties.

U_R	U_M	U_C	U_A
0 : 1.00	0 : 1.00	0 : 1.00	0 : 1.00
100 : 1.00	5 : 1.00	1 : 0.90	100 : 0.00
200 : 0.99	20 : 0.80	2 : 0.30	
500 : 0.90	50 : 0.40	3 : 0.10	
1000 : 0.75	70 : 0.00		
1500 : 0.50			
2000 : 0.25			
4000 : 0.00			

Table 3: Utility functions for Znn.com.

In this case, utility functions are defined by an explicit set of value pairs (with intermediate points linearly interpolated). Table 3 summarizes the utility functions for Znn.com. Function U_R maps low response times (up to 100 ms) with maximum utility, whereas values above 2000 ms are highly penalized (utility below 0.25), and response times above 4000 ms provide no utility. In this case, utility and mapped property values across all quality dimensions are inversely proportional, although this is not necessarily true in general.

Scenario	**Priority**	w_{U_R}	w_{U_M}	w_{U_C}	w_{U_A}
1	Minimizing number of malicious clients.	0.15	0.6	0.1	0.15
2	Optimizing good client experience.	0.3	0.3	0.1	0.3

Table 4: Utility preferences for Znn.com scenarios.

Utility preferences capture business preferences over the quality dimensions, as-

signing a specific weight to each one of them. We consider two scenarios in Znn.com, whose priority concerns are summarized in Table 4.

4.2. Adaptation Strategy Selection

A situation that demands adaptation can generally be addressed in different ways by executing alternative adaptation strategies, many of which may be applicable under the same runtime conditions. Different strategies impact run time quality attributes in various ways, thus there is a need to choose a strategy that will result in the best outcome with respect to achieving the system's desired quality objectives.

To enable decision-making for selecting strategies we use utility functions and preferences, which are sensitive to the context of use and able to consider trade-offs among multiple potentially conflicting objectives. By evaluating all applicable strategies against the different quality objectives, we obtain an aggregate expected utility value for each strategy by using the specified utility preferences. The strategy selected for execution by the adaptation manager is the one that maximizes expected utility.

The aggregated impact on utility of a strategy is obtained by: (i) computing the aggregate impact of the strategy on the system's state, (ii) merging aggregated strategy impact with current system state to obtain the expected state after strategy execution, (iii) mapping expected state to utilities, and (iv) combining all utilities using utility preferences.

As an example of how the utility of a strategy is calculated, let us assume that the adaptation cycle is triggered in system state [1500, 90, 2, 0], indicating response time, percentage of malicious clients, operating cost, and user annoyance level, respectively. We focus on the evaluation of strategy Outgun.

To obtain the aggregate impact on system state of a strategy, we need to estimate the likelihood of selecting different tactics at runtime due to the uncertainty in their selection and outcome within the strategy tree. To this end, Rainbow uses a stochastic model of a strategy, assigning a probability of selection to every branch in the tree (by default, divided equally among the branches). Figure 2 shows how the aggregate impact on state is computed bottom-up in the strategy tree: the aggregate impact of each node is computed by adding the aggregate impact of its children, reduced by the

probability of their respective branches, with the cost-benefit attribute vector of the tactic in the node (if any). In the example, the impact contributed by nodes t0 and t2 correspond to the cost-benefit vectors of the associated tactics, whereas leaf nodes make no changes to the system and therefore have no impact.

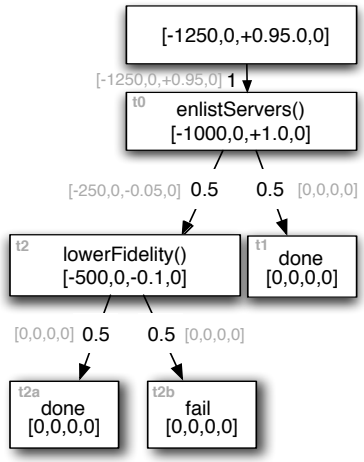

Figure 2: Calculation for aggregate impact of strategy Outgun.

Once the aggregate impact of the strategy is computed, it is merged with the current system state to obtain the expected system state after strategy execution:

$$[1500,90,2,0]+[-1250,0,+0.95,0]=[250,90,2.95,0]$$

Next, we map the expected conditions to the utility space:

$$[U_R(250),U_M(0),U_C(2.95),U_A(0)]=[0.975, 1.0, 0.11, 1.0]$$

And finally, all utilities are combined into a single utility value by making use of the utility preferences. Hence, if we assume that we are in scenario 2, the aggregate utility for strategy Outgun would be:

$$0.975*0.3+1.0*0.3+0.11*0.1+1.0*0.3=0.9035$$

Utility scores are computed similarly for all strategies. In this case, strategies Eliminate and Outgun score 0.81 and 0.90 respectively, thus Outgun would be selected.

5. Analyzing Trade-Offs of Involving Humans in Adaptation

In the previous section, we described a language to express adaptation models that can be analyzed to evaluate trade-offs among different concerns in self-adaptation. In

this section, we incorporate a model of the human operators interacting with the system to extend the language. This extension enables the evaluation of trade-offs of involving humans in adaptation with respect to a given set of concerns and preferences expressed in a utility profile.

5.1. 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 [6]. These models extend the description of the underlying system's architecture, and 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:

5.1.1. Opportunity

captures the applicability conditions of the adaptation tactics that can be executed by human actors on 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 operator manually select malicious clients to blackhole (*blackholeAttacker*) in a DoS attack scenario. Opportunity elements are* $OE^{\text{blackholeAttacker}} = \{L, B\}$*, 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 $OE^{\text{blackholeAttacker}}$*, an opportunity function for the tactic* $f_O^{\text{blackholeAttacker}} = (L.state == ONL) \cdot (B.state == ONB)$ *can be used to constrain its applicability only to situations in which there is an operator on location who is not busy.*

5.1.2. 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 blackholeAttacker tactic can be defined as $WE^{\text{blackholeAttacker}} = \{S\}$, 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 $f_W^{\text{blackholeAttacker}} = pr_W(S.state)$, with $pr_W : S \rightarrow [0, 1]$.

5.1.3. 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 $CE^{\text{blackholeAttacker}} = \{T\}$, 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 $f_C^{\text{blackholeAttacker}} = pr_C(T.state)$, with $pr_C : T \rightarrow [0, 1]$. This models the fact that better trained operators are more effective at eliminating malicious users and less likely to blackhole legitimate clients, resulting in better reductions in the percentage of malicious clients with no or little increments in the level of user annoyance.

5.2. Integrating Human and Adaptation Models

Incorporation of OWC elements for adaptation execution in Stitch affects the specification of different elements in adaptation tactics and strategies.

5.2.1. Tactics

In tactics involving humans, constraints that affect the applicability of a tactic can be derived either from the human model (opportunity elements), or properties of the architecture itself (e.g., are there any available servers to activate?). In general, applicability conditions of these tactics will be a combination of both. In Listing 3, the condition block of tactic blackholeAttacker (line 4) combines opportunity elements from the human model (operator on location and not busy – predicate ONLNB, defined in line 1), with a predicate defined over the properties of the architecture (legitimate clients are experiencing a high response time – cHiRespTime).

The action block of these tactics can execute automated operations, as in the case of tactic enlistServers (Listing 1), but also notify human actors to perform a task. The action block of of tactic blackholeAttacker (Listing 3, lines 5-6) first selects an available operator (line 5), and next it notifies the selected operator that she has to blackhole potentially malicious clients via a text message (line 6).

```
1 define boolean ONLNB=exists o:operatorT in M.participants | o.onLocation && !o.busy;
2 ...
3 tactic blackholeAttacker(){
4   condition {ONLNB && cHiRespTime;}
5   action {ao=Set.RandomSubSet({select o:operatorT in M.participants | o.onLocation && !o.busy},1);
6           notify(ao, "Blackhole potentially malicious clients");}
7   effect {!cHiRespTime;}
8 }
```

Listing 3: Tactic for blackholing malicious clients via human operator.

5.2.2. Strategies

Fully automated, as well as tactics involving humans can be combined to achieve better outcomes in adaptation strategies. Listing 4 shows strategy Eliminate for eliminating excess traffic from potentially malicious clients first by notifying an operator (via tactic blackholeAttacker, line 5) to manually block traffic from malicious clients. If the assigned time window of 5 minutes expires and the intended effect of the tactic (!cHiRespTime, Listing 3) is not observed, the strategy notifies an operator to execute the throttleSuspicious tactic as a fallback, throttling suspicious clients (line 7).

6. Reasoning about Human-in-the-Loop Adaptation

When defining a collection of adaptation strategies and their associated utility profile, we need to guarantee not only that the system will carry out reasonable choices under all possible circumstances, but also that the effect of those choices combined with the behavior of human participants will have a reasonable impact on business concerns. To provide such guarantees, we make use of a formal model based on an abstraction of the extended Stitch profile for human-in-the-loop adaptation presented

```
1  define boolean unhandledMalicious=exists c:T.ClientT in M.components | c.maliciousness>M.MAL_THR && !c.isBlackHoled;
2  define boolean unhandledSuspicious=exists c:T.ClientT in M.components | c.maliciousness > M.SUS_THR and !c.isThrottled;
3  ...
4  strategy Eliminate [unhandledMalicious || unhandledSuspicious] {
5    t0: (unHandledMalicious) -> blackholeAttacker () @[300000] {
6        t0a: (success) -> done;
7        t0b: (unHandledSuspicious) -> throttleSuspicious () @[300000] {
8             t1a: (success) -> done;
9         t1b: (default) -> fail; }
10   }
11 }
```

Listing 4: Strategy to eliminate excess traffic in Znn.com

in Section 5.1 that enables us to reason about: (i) the choices made by the adaptation manager for adaptation strategy selection, and (ii) the impact of the execution of selected adaptation strategies on the target system.

Our modeling approach for human-in-the-loop adaptation consists in describing a stochastic multiplayer game in which we consider that one of the players is the adaptive system (including both automated adaptation mechanisms and human actors) and the other is the environment within which the system operates. The goal of the system player is to maximize accrued utility during the system's execution (encoded formally as a reward structure), while we consider the environment to be an antagonistic player that tries to minimize that same reward.

In the remainder of this section, we first introduce some background on model checking SMGs, the formal technique that we use to formally reason about human involvement in adaptation. Next, we provide a description of our Znn.com model implemented in the probabilistic model-checker PRISM-games [19], as well as the analysis and results that we obtained for human-in-the-loop adaptation analysis.

6.1. Model Checking Stochastic Multiplayer Games

Automatic verification techniques for probabilistic systems 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, or time).

Competitive behavior may also appear in (stochastic) 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. Automatic verification techniques have been successfully used in this context, for instance for the analysis of security [20] or communication protocols [21].

Our approach to analyzing human involvement in adaptation builds upon a recent technique for modeling and analyzing SMGs [22]. 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. Players can either cooperate to achieve the same goal, or compete to achieve their own goals.

The approach includes a logic called rPATL for expressing quantitative properties of stochastic multiplayer games, which extends the probabilistic logic PATL [23]. PATL is itself an extension of ATL [24], a logic extensively used in multiplayer games and multiagent 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[3] which can ensure that either 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 $\mathsf{P}_{\bowtie q}$ and path formulae from PCTL [25]. Moreover, rPATL includes a generalization of the reward operator $\mathsf{R}^{r}_{\bowtie x}$ from [26] to reason about goals related to rewards. An extended version of the rPATL reward operator $\langle\langle C \rangle\rangle \mathsf{R}^{r}_{\mathsf{max}=?}[\mathsf{F}^{\star}\ \phi]$ [4] 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

[3]The term *strategy* employed in the context of SMGs refers to a *game strategy* (referred to also as *policy* or *adversary*) as defined in [22], and should not be confused with Stitch adaptation strategies.

[4]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 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."

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 including an extended version of the rPATL reward operator. The checking of such properties also supports strategy synthesis, enabling us to obtain the corresponding optimal strategy. An SMG strategy resolves the choices in each state, selecting actions for a player based on the current state and a set of memory elements.[5]

6.2. Formal Model

Our game is played in turns by two players that are in control of the behavior of the environment and a Znn.com system (including human actors), respectively. The SMG model consists of the following parts:

6.2.1. Player definition

Listing 5 illustrates player definition in the SMG. Player env is in control of all the (asynchronous) actions that the environment can take (defined in the environment module), while system player sys controls all the actions that belong to the human actor and the target system, whose behavior is specified in the processes ha_system, as well as Outgun and Eliminate (adaptation strategies for absorbing and eliminating excess traffic, respectively). Moreover, the system player controls the synchronization of actions between adaptation strategies and the target system, thus modeling the triggering of adaptation tactics. Global variable turn (line 4) is used to explicitly encode alternating turns between the system and environment players.

6.2.2. Environment

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 our model a simple behavior

[5] See [22] for more details on SMG strategy synthesis.

```
1  player sys ha_system, Eliminate, Outgun, [blackholeAttacker], [throttleSuspicious], [enlistServers],
       [lowerFidelity] endplayer
2  player env environment endplayer
3  const ENVT=0; const SYST=1;
4  global turn:[ENVT..SYST] init ENVT;
```

Listing 5: Player definition for the Znn.com SMG.

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 [16] for further details illustrating the modeling of adversarial environment behavior in turn-based SMGs).

```
1  const MAX_TIME; // Exercution time frame [0,MAX_TIME]
2  module environment
3    t:[0..MAX_TIME] init 0;
4    [] (turn=ENVT) & (t<MAX_TIME) -> (t'=t+1) & (turn'=SYST);
5  endmodule
```

Listing 6: Environment module.

6.2.3. Human Model

Listing 7 shows the encoding of the OWC elements corresponding to an operator in the Znn.com system. Opportunity elements (line 2) indicate whether the operator is on location and/or busy. These predicates are used to guard the execution of tactics blackholeAttacker and throttleSuspicious in the model (Listing 8, line 19). The willingness function of the operator (line 6) is inversely proportional to her stress level, declared in line 5. The capability function (line 9) corresponds to the training level of the operator in this case.

6.2.4. System

The combined behavior of the target system and human actors is described in module ha_system (Listing 8). The module incorporates a collection of variables encoding the different system qualities of concern, as well as the aspects relevant to the applicability conditions of tactics (e.g., values of predicates used in the condition block of a tactic). Lines 12-17 illustrate how the different variables are initialized:

```
1  // Opportunity elements
2  global op_onLocation:bool init true, op_busy: bool init false;
3  // Willingness elements and function
4  const MAX_STRESS_LEVEL, INIT_STRESS_LEVEL;
5  global op_stressLevel: [0..MAX_STRESS_LEVEL] init INIT_STRESS_LEVEL;
6  formula op_f_w=(MAX_STRESS_LEVEL−op_stressLevel) / MAX_STRESS_LEVEL;
7  // Capability elements and function
8  const double op_trainingLevel;
9  formula op_f_c= op_trainingLevel;
10 // Combined WC probability for tactic BlackholeAttacker
11 formula ba_wc_prob = op_f_c ∗ op_f_w;
```

Listing 7: Human actor model encoding for a Znn.com operator.

- rt, as, mc, and ua encode the response time, number of active servers, percentage of malicious clients, and level of user annoyance in the system, respectively.

- cnt_es and cnt_ba are counters used to keep track of the latency of tactics enlistServers and blackholeAttacker, respectively.[6]

Moreover, the module includes commands that model the effect of executing the different tactics as updates on its variables. In particular, there are three different commands per tactic in the module. We focus on tactic blackholeAttacker to illustrate how tactic execution is modeled:

- *Tactic trigger* (line 19). Triggers tactic execution when: (i) an operator is on location and not busy, (ii) the estimated percentage of malicious clients is above zero, 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 (cnt_ba'=1).

- *Tactic latency counter update* (line 22). If the tactic counter is active, but still has not reached the tactic's latency value, the counter is incremented in one unit.

- *Tactic completion* (line 25). When the tactic's latency counter expires, the command updates variables rt, mc and ua according to the encoding of the impact of

[6] We do not describe the code corresponding to tactics lowerFidelity and throttleSuspicious in Listing 8 for the sake of clarity.

the tactics on the different quality dimensions (lines 5-7), which are affected by the probability ba_wc_prob (determined by the willingness and capability elements defined in Listing 7). The latency counter is reset, and the busy status of the operator is set to false.

The encoding used for the enlistServers tactic (lines 20,23,26) follows the same structure, but without any OWC elements encoded in the guards or updates of the commands.

Every command in the module includes a predicate in the guard to ensure that the command is triggered only during the system player's turn (turn=SYST), and an additional predicate in the post state that yields the turn to the environment player (turn'=ENVT). Moreover, an additional command (line 28) lets the process progress without any variable updates when none of the latency periods for the tactics are active. Note that in our model, we assume sequential execution of tactics (in accordance with Stitch semantics).

6.2.5. Adaptation logic

Modules Eliminate and Outgun model the adaptation logic placed in the controller, according to the description of their respective Stitch strategies described in Listings 4 and 2. Each of the commands corresponds to a tactic that can be executed in the target system via synchronization on shared action names with trigger commands in the ha_system module (Listing 8, lines 19-20).

Module Eliminate (Listing 9) models the strategy to eliminate excess traffic with the help of a human operator. The command on line 3 encodes the triggering of tactic blackholeAttacker [7], which sets the value of the timestamp variable ba_trigger_t that indicates at which time point the tactic was triggered. This variable is used on the guard of the command encoding the execution of throttleSuspicious (line 4) to determine whether the settling time for observation of the previous tactic's effect has already expired. If this is the case, and the blackholing of malicious clients by the human operator

[7] We abstract away predicates unhandledMalicious and unhandledSuspicious (Listing 4, lines 4,5), which we assume to be true in the scenarios encoded in our model.

```
1  // EnlistServer Tactic cost/benefit attribute vector functions
2  formula es_f_rt = rt−1000 >=0 ? (rt−1000<=MAX_RT? rt−1000 : MAX_RT) : 0;
3  formula es_f_as=as<MAX_SERVERS ? as+1:as;
4  // BlackholeAttacker Tactic cost/benefit attribute vector functions
5  formula ba_f_rt = rt−1000 >=0 ? (rt−1000<=MAX_RT? rt−1000 : MAX_RT) : 0;
6  formula ba_f_mc = ba_wc_prob∗mc > 0 ? (ba_wc_prob∗mc < 100? floor(ba_wc_prob∗mc) : 100) : 0;
7  formula ba_f_ua = ua+(1−ba_wc_prob)∗50 >=0 ? (ua+(1−ba_wc_prob)∗50<=100? floor(ua+(1−ba_wc_prob)∗50)
       : 100) : 0;
8  ...
9  formula cost=as ∗ cost_per_server;
10 ...
11 module ha_system
12 rt : [0..MAX_RT] init init_rt; // Response time
13 as : [0..MAX_SERVERS] init init_as; // Active servers
14 mc : [0..100] init init_mc; // Malicious clients
15 ua : [0..100] init init_ua; // level of annoyance
16 cnt_es :[0..MAX_TIME] init 0;
17 cnt_ba :[0..MAX_TIME] init 0;
18 // Tactic triggers
19 [blackholeAttacker] (turn=SYST) & (op_onLocation) & (!op_busy) & (mc>0) & (cnt_ba=0) −> (cnt_ba'=1) &
       (op_busy'=true);
20 [enlistServers] (turn=SYST) & (as<MAX_SERVERS) & (cnt_es=0) −> (cnt_es'=1) & (turn'=ENVT);
21 // Tactic latency counter update
22 [] (turn=SYST) & (cnt_ba>0) & (cnt_ba<ba_latency) −> (cnt_ba'=cnt_ba+1) & (turn'=ENVT);
23 [] (turn=SYST) & (cnt_es>0) & (cnt_es<es_latency) −> (cnt_es'=cnt_es+1) & (turn'=ENVT);
24 // Tactic completion (after latency period expires)
25 [] (turn=SYST) & (cnt_ba=ba_latency) −> (cnt_ba'=0) & (rt'=ba_f_p) & (mc'=ba_f_mc) & (ua'=ba_f_ua) &
       (op_busy'=false) & (turn'=ENVT);
26 [] (turn=SYST) & (cnt_es=es_latency) −> (rt'=es_f_rt) & (cnt_es'=cnt_es+1) & (as'=es_f_as) & (cnt_es'=0) &
       (turn'=ENVT);
27 // Do nothing
28 [] (turn=SYST) & (cnt_es=0) & ... & (cnt_ba=0) −> (turn'=ENVT);
29 endmodule
```

Listing 8: Target system extended with human actors module.

is not successful (ba_fail), the command executes, triggering the throttleSuspicious tactic, consistently with the Stitch code in listing 4, line 4.

Module Outgun (Listing 9, line 7) follows a similar PRISM encoding that models the automatic strategy to absorb excess traffic in Znn.com.

```
1  module Eliminate
2  ba_trigger_t:[0..MAX_TIME] init 0;
3  [blackholeAttacker] (turn=SYST) -> (ba_trigger_t'=t);
4  [throttleSuspicious] (turn=SYST) & (t=ba_trigger_t+ba_settling) & (ba_fail) -> true;
5  endmodule
6
7  module Outgun
8  es_trigger_t:[0..MAX_TIME] init 0;
9  [enlistServers] (turn=SYST) -> (es_trigger_t'=t);
10 [lowerFidelity] (turn=SYST) & (t=es_trigger_t+es_settling) & (es_fail) -> true;
11 endmodule
```

Listing 9: Eliminate and Outgun adaptation strategy modules.

6.2.6. Utility Profile

Utility functions and preferences are encoded using formulas and reward structures that enable the quantification of the utility of a given game state. Formulas compute utility on the different dimensions of concern, and reward structures weigh them against each other by using the utility preferences of a given scenario.

```
1  formula uM = (mc>=0 & mc <=5? 1:0)
2    +(mc>5 & mc <=20? 1+(0.80-1)*((mc-5)/(20-5)):0)
3    +(mc>20 & mc <=50? 0.80+(0.40-0.80)*((mc-20)/(50-20)):0)
4    +(mc>50 & mc <=70? 0.40+(0.00-0.40)*((mc-50)/(70-50)):0)
5    +(mc>70 ? 0:0);
6  ...
7  rewards "rGU"
8   scenario=1 : 0.15*uR +0.6*uM +0.1*uC +0.15*uA;
9  ...
10 endrewards
```

Listing 10: Utility reward structure for Znn.com DoS scenarios.

Listing 10 illustrates in lines 1-5 the encoding of utility functions using a formula for linear interpolation based on the points defined for utility function U_M in the second column of Table 3. Lines 7-10 show how a reward structure can be defined to compute a single utility value for any state by using the utility preferences defined for a particular scenario.

6.3. Analysis

SMG models of human-in-the-loop adaptation can be exploited to determine: (i) the expected outcome of human involvement in adaptation, and (ii) the conditions under which such involvement improves over fully automated adaptation. To answer these questions, we make use of rPATL specifications that include reward-specific operators aimed at checking quantitative properties over SMG models. Specifically, our technique enables us to statically analyze a particular region of the state space, which first has to be discretized to check rPATL properties. Obtaining the results of the analysis for each state in the discrete set requires an independent run of the model checker in which model parameters are instantiated with variable values corresponding to that state.

6.3.1. Strategy Utility

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

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

The property obtains the maximum accrued utility value (i.e., corresponding to reward rGU – Listing 10) that the system player can achieve until the end of execution (t=MAX_TIME).

Figure 3(a) depicts strategy utility analysis results for the different adaptation strategies in a DoS scenario in which the priority is eliminating malicious clients (Scenario 1 in Table 4). In the figure, a discretized region of the state space is projected over the dimensions that correspond to the training level of a human actor, and the percentage of malicious clients (with values in the range [0,1] and [0,100], respectively). Each point in the mesh represents the maximum accrued utility that the system can achieve on the model instanced for a time frame of 15 minutes. The initial state of the scenario corresponds to 0 stress level of the operator, a response time is 2000 ms, 0% of user annoyance, and 2 active servers. Tactic cost/benefit values and the utility profile employed are those described in Section 4, whereas the latency value employed for tactics blackholeAttacker and throttleSuspicious is 5 minutes (this latency models the time that the

human operator takes to decide which clients have to be blackholed or throttled). Time delays to observe the effect of tactic executions in the different strategies are those indicated in the Stitch code shown in Listings 4 and 2, respectively.

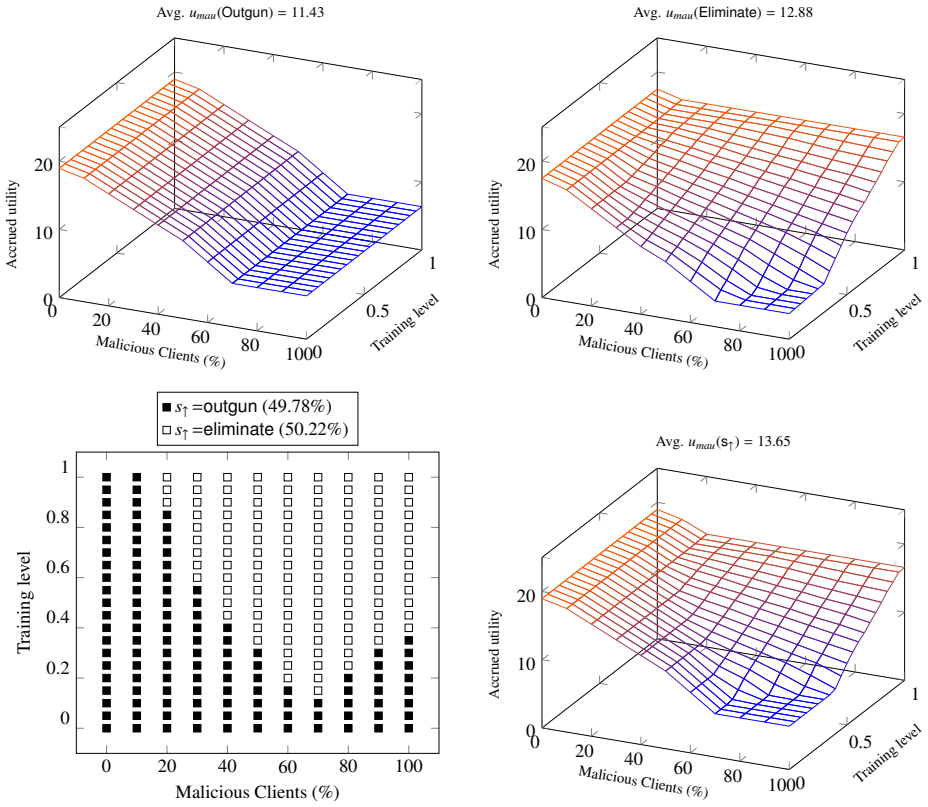

Figure 3: Results for Scenario 1 (minimizing number of malicious clients): (a) outgun (top left) and eliminate (top right) strategy utility, (b) strategy selection (bottom left), and (c) combined utility (bottom right).

In the top left of Figure 3, the plot shows that the utility obtained by the strategy Outgun in this scenario is not affected by the level of training of the human operator because the tactics employed by the strategy are fully automated. Moreover, the utility that can be obtained decreases progressively with increasing levels of malicious clients. This is consistent with the fact that strategy Outgun employs only tactics that try to improve user experience without dealing with malicious users (e.g., adding new servers), and in Scenario 1, the main contribution to utility results from low levels of malicious

clients.

The top right of Figure 3 depicts the utility obtained by strategy Eliminate. In contrast with strategy Outgun, the plot shows how increasing levels of training yield better results. When the percentage of malicious clients is low, the impact of training on utility is negligible because there are few or no malicious clients to deal with. However, the outcome of the execution of tactics blackholeAttacker and throttleSuspicious in situations with increasing levels of malicious clients can vary significantly depending on the level of training of the human operator, who has to judge which clients will be affected by the tactics. Poorly trained operators can often apply counter measures to legitimate clients, being less efficient at reducing the percentage of malicious clients while increasing the level of annoyance in legitimate clients when blackholing or throttling.

Figure 4 shows results for Scenario 2, in which the top priority is optimizing the experience of legitimate clients, independently of the level of malicious clients making use of system resources. The top left plot of the figure shows how strategy Outgun still experiences a reduction in the utility with increasing levels of malicious users (similarly to Scenario 1). However, in this scenario the reduction in utility is less pronounced than in Scenario 1 because in this case the main contribution to utility results from optimizing legitimate client experience, and efficiency at reducing the percentage of malicious clients is not as relevant.

6.3.2. 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 3(b) shows the results of the analysis of strategy selection in Scenario 1. The states in which human involvement via strategy Eliminate is chosen (50.22% of states)

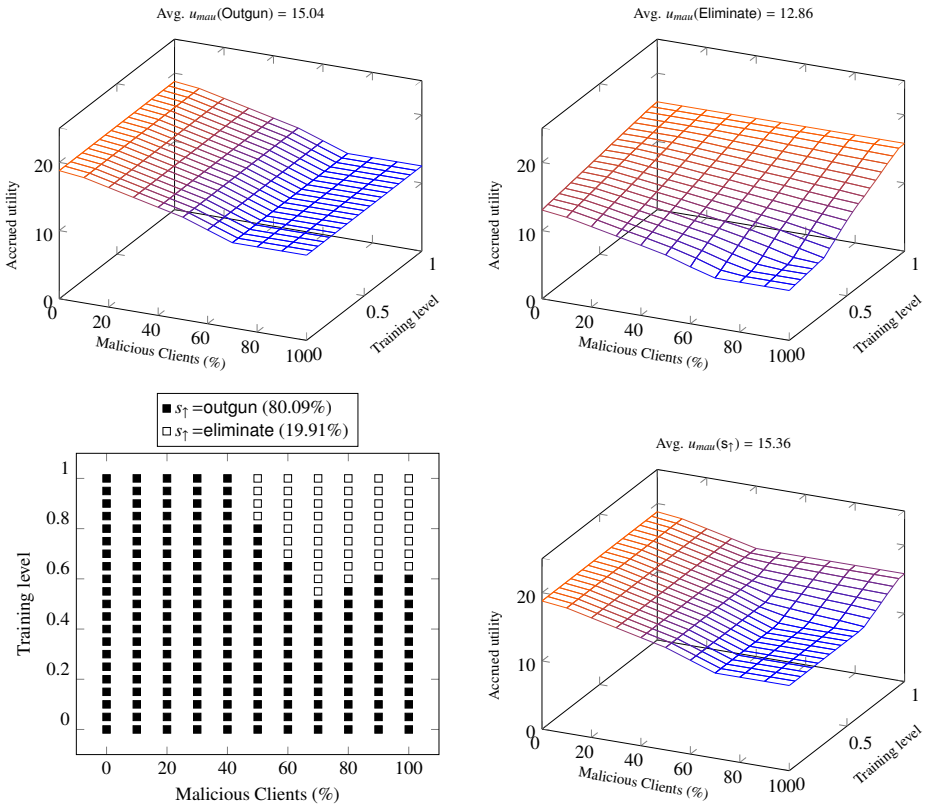

Figure 4: Results for Scenario 2 (optimizing experience of legitimate clients): (a) outgun (top left) and eliminate (top right) strategy utility, (b) strategy selection (bottom left), and (c) combined utility (bottom right).

are represented in white, whereas states in which the automated strategy Outgun is selected (49.78%) are colored in black. The figure shows how progressively higher levels of malicious clients make human involvement preferable even when the level of training of the operator is limited (0.3-0.4) because, even under these conditions, Eliminate is still better at improving utility than Outgun. This is explained by the fact that the top priority in Scenario 1 is minimizing the number of malicious clients, and Outgun does not employ any tactics for dealing with them. However, it is worth noting that when the training level is very low, the improvement on user experience provided by Outgun can outweigh the moderate improvement in utility provided by inefficient exe-

cutions of Eliminate (even if the percentage of malicious clients is high). This situation can be observed in the area in which training levels are below 0.4 and the percentage of malicious clients are in the range 80-100%.

Figure 3(c) shows the combined accrued utility mesh that results from the selection process (i.e, every point in the mesh is computed as $u_{mau}(s_{\uparrow})$). The average improvement is 16.3% over the Outgun strategy, and 5.6% over Eliminate. 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.

Figure 4(b) shows the results of the analysis of strategy selection in Scenario 2. In this case, the plot shows how Outgun is selected in more than 80% of the states. This represents a remarkable increment in the selection of the automated strategy with respect to Scenario 1, which is explained by the different priorities that exist in Scenario 2 (improving legitimate client experience, independently of the percentage of malicious clients). Indeed, it can be observed that the selection of Eliminate in this scenario is justified only in the area in which both the percentage of malicious clients and the training level of the operator are high.

Figure 4(c) shows the combined accrued utility mesh in Scenario 2. In this case, the improvement in utility obtained by the combined approach with respect to the individual strategies is not too far from those in Scenario 1, but transposed (the improvement over Outgun is 2%, and 16.2% for Eliminate). This is motivated by the better alignment of the priorities in Scenario 2 and the target of strategy Outgun (improving client experience), whereas the priorities of Scenario 1 are better aligned with the target of Eliminate (dealing with malicious clients).

7. Conclusion

In this chapter, we have described an approach that employs formal reasoning to analyze trade-offs in self-adaptation at two different levels: (i) reasoning about business concerns in the context of other (potentially conflicting) business properties; and (ii) reasoning about the effectiveness of automated *vs.* human-driven adaptations with

respect to the different business concerns.

We have focused on the execution stage of MAPE-K systems, in which human actors adopt the role of system-level effectors. We have shown how to incorporate concepts from cyber-human systems (CHS) that model the probabilistic aspects of human behavior into a language tailored to describe runtime adaptation (Stitch). We have also shown how such specifications can be encoded into stochastic multiplayer game models amenable to analysis via model checking. We illustrated our approach in the context of Znn.com, a benchmark system in the self-adaptive systems community that embodies the typical infrastructure of a dynamically scalable web infrastructure. Our results showed that our approach can: (i) discriminate cases in which the involvement of human actors in execution leads to an improvement of system utility, providing the basis to combine human-based and automated adaptations; and (ii) decide about human involvement in a context-sensitive manner, selecting different adaptations for different preferences over business concerns.

Concerning future work, 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 [6]. To capture this subtlety, we plan on extending the encoding of SMGs to model human actors as separate players. Moreover, we will extend our 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.

Acknowledgment

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-XXXXXXX).

References

[1] 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, J. Whittle, Software engineering for self-adaptive systems: A research roadmap, in: B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee (Eds.), Software Engineering for Self-Adaptive Systems, Vol. 5525 of Lecture Notes in Computer Science, Springer, 2009, pp. 1–26.

[2] M. C. Huebscher, J. A. McCann, A survey of autonomic computing - degrees, models, and applications, ACM Comput. Surv. 40 (3).

[3] J. O. Kephart, D. M. Chess, The vision of autonomic computing, IEEE Computer 36 (1) (2003) 41–50.

[4] D. Garlan, S.-W. Cheng, A. Huang, B. Schmerl, P. Steenkiste, Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure, IEEE Computer 37 (10) (2004) 46–54.

[5] S.-W. Cheng, D. Garlan, Stitch: A language for architecture-based self-adaptation, Journal of Systems and Software 85 (12) (2012) 2860–2875.

[6] D. Eskins, 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, IEEE Computer Society, 2011, pp. 233–242.

[7] S. Cheng, et al., Evaluating the Effectiveness of the Rainbow Self-Adaptive System, in: SEAMS, 2009.

[8] N. Esfahani, S. Malek, Uncertainty in self-adaptive software systems, in: R. de Lemos, H. Giese, H. Muller, M. Shaw (Eds.), Software Engineering for Self-Adaptive Systems II, Vol. 7475 of Lecture Notes in Computer Science, Springer, 2013, pp. 214–238.

[9] D. Sykes, W. Heaven, J. Magee, J. Kramer, Exploiting non-functional preferences in architectural adaptation for self-managed systems, in: S. Y. Shin, S. Ossowski, M. Schumacher, M. J. Palakal, C. Hung (Eds.), Proceedings of the 2010 ACM Symposium on Applied Computing (SAC), Sierre, Switzerland, March 22-26, 2010, ACM, 2010, pp. 431–438.

[10] R. Calinescu, M. Z. Kwiatkowska, Using quantitative analysis to implement autonomic IT systems, in: J. M. Atlee, P. Inverardi (Eds.), 31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings, IEEE, 2009, pp. 100–110.

[11] I. Epifani, C. Ghezzi, R. Mirandola, G. Tamburrelli, Model evolution by run-time parameter adaptation, in: J. M. Atlee, P. Inverardi (Eds.), 31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings, IEEE, 2009, pp. 111–121.

[12] P. Oreizy, M. Gorlick, R. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, A. Wolf, An architecture-based approach to self-adaptive software, Intelligent Systems and their Applications, IEEE 14 (3) (1999) 54–62.

[13] C. Dorn, R. N. Taylor, Coupling software architecture and human architecture for collaboration-aware system adaptation, in: D. Notkin, B. H. C. Cheng, K. Pohl (Eds.), 35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013, IEEE / ACM, 2013, pp. 53–62.

[14] L. Clement, D. Konig, V. Mehta, R. Mueller, R. Rangaswamy, M. Rowley, I. Trickovic, WS-BPEL extension for people BPEL4People, http://docs.oasis-open.org/bpel4people/bpel4people-1.1.html (2010).

[15] P. Wohed, W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, N. Russell, On the suitability of BPMN for business process modelling, in: S. Dustdar, J. L. Fiadeiro, A. P. Sheth (Eds.), Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings, Vol. 4102 of Lecture Notes in Computer Science, Springer, 2006, pp. 161–176.

[16] J. Cámara, G. A. Moreno, D. Garlan, Stochastic game analysis and latency awareness for proactive self-adaptation, in: G. Engels, N. Bencomo (Eds.), 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2014, Proceedings, Hyderabad, India, June 2-3, 2014, ACM, 2014, pp. 155–164.

[17] J. Kramer, J. Magee, Self-managed systems: an architectural challenge, in: L. C. Briand, A. L. Wolf (Eds.), International Conference on Software Engineering, ISCE 2007, Workshop on the Future of Software Engineering, FOSE 2007, May 23-25, 2007, Minneapolis, MN, USA, 2007, pp. 259–268.

[18] J. Cámara, A. Lopes, D. Garlan, B. 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, Vol. 8997 of Lecture Notes in Computer Science, Springer, 2014, pp. 89–107.

[19] T. Chen, V. Forejt, M. Z. Kwiatkowska, D. Parker, A. Simaitis, Prism-games: A model checker for stochastic multi-player games, in: N. Piterman, S. A. Smolka (Eds.), Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings, Vol. 7795 of Lecture Notes in Computer Science, Springer, 2013, pp. 185–191.

[20] S. Kremer, J. Raskin, A game-based verification of non-repudiation and fair exchange protocols, in: K. G. Larsen, M. Nielsen (Eds.), CONCUR 2001 - Concurrency Theory, 12th International Conference, Aalborg, Denmark, August 20-25, 2001, Proceedings, Vol. 2154 of Lecture Notes in Computer Science, Springer, 2001, pp. 551–565.

[21] W. V. D. Hoek, M. Wooldridge, Model checking cooperation, knowledge, and time - a case study, in: Research in Economics, 2003.

[22] T. Chen, V. Forejt, M. Z. Kwiatkowska, D. Parker, A. Simaitis, Automatic verification of competitive stochastic systems, Formal Methods in System Design 43 (1) (2013) 61–92.

[23] T. Chen, J. Lu, Probabilistic alternating-time temporal logic and model checking algorithm, in: J. Lei (Ed.), Fourth International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2007, 24-27 August 2007, Haikou, Hainan, China, Proceedings, Volume 2, IEEE Computer Society, 2007, pp. 35–39.

[24] R. Alur, T. A. Henzinger, O. Kupferman, Alternating-time temporal logic, Journal of the ACM 49 (5) (2002) 672–713.

[25] A. Bianco, L. de Alfaro, Model checking of probabalistic and nondeterministic systems, in: P. S. Thiagarajan (Ed.), Foundations of Software Technology and Theoretical Computer Science, 15th Conference, Bangalore, India, December 18-20, 1995, Proceedings, Vol. 1026 of Lecture Notes in Computer Science, Springer, 1995, pp. 499–513.

[26] V. Forejt, M. Z. Kwiatkowska, G. Norman, D. Parker, Automated verification techniques for probabilistic systems, in: M. Bernardo, V. Issarny (Eds.), Formal Methods for Eternal Networked Software Systems - 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures, Vol. 6659 of Lecture Notes in Computer Science, Springer, 2011, pp. 53–113.