Reasoning about Human Participation in Self-Adaptive Systems

Javier Cámara
Institute for Software Research
Carnegie Mellon University
Pittsburgh, PA 15213, USA
Email: jcmoreno@cs.cmu.edu

Gabriel A. Moreno
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
Email: gmoreno@sei.cmu.edu

David Garlan
Institute for Software Research
Carnegie Mellon University
Pittsburgh, PA 15213, USA
Email: garlan@cs.cmu.edu

***Abstract*—Self-adaptive systems overcome many of the limitations of human supervision in complex software-intensive systems by endowing them with the ability to automatically adapt their structure and behavior in the presence of runtime changes. However, adaptation in some classes of systems (e.g., safety-critical) can benefit by receiving information from humans (e.g., acting as sophisticated sensors, decision-makers), or by involving them as system-level effectors to execute adaptations (e.g., when automation is not possible, or as a fallback mechanism). However, human participants are influenced by factors external to the system (e.g., training level, fatigue) that affect the likelihood of success when they perform a task, its duration, or even if they are willing to perform it in the first place. Without careful consideration of these factors, it is unclear how to decide when to involve humans in adaptation, and in which way. In this paper, we investigate how the explicit modeling of human participants can provide a better insight into the trade-offs of involving humans in adaptation. We contribute a formal framework to reason about human involvement in self-adaptation, focusing on the role of human participants as actors (i.e., effectors) during the execution stage of adaptation. The approach consists of: (i) a language to express adaptation models that capture factors affecting human behavior and its interactions with the system, and (ii) a formalization of these adaptation models as stochastic multiplayer games (SMGs) that can be used to analyze human-system-environment interactions. We illustrate our approach in an adaptive industrial middleware used to monitor and manage sensor networks in renewable energy production plants.**

I. Introduction

Software-intensive systems are increasingly relied upon to support a wide variety of tasks in modern society. Still, the ability of these systems to provide service that can be trusted in the presence of changes, either in their environment (e.g., resource availability) or the system itself (e.g., faults) is increasingly affected by their growing complexity, as well as by the dynamic and unpredictable nature of the environments in which they typically have to operate.

Early attempts to address this situation involved human supervision, which is expensive and unreliable, due to the fallible nature of humans and their difficulty to react in a timely manner.

Over the last decade, self-adaptive systems [11], [21], [22] have emerged as an alternative that overcomes many of the limitations of human supervision by endowing systems with mechanisms to automatically adapt their structure and behavior at run time. Self-adaptation approaches usually rely on closed-loop controllers (e.g., implementing the MAPE-K model [22]) that eliminate human intervention from adaptation. Although these fully automated approaches have proven successful in many application domains, there are situations in which they may be suboptimal, or simply do not suffice to effect changes at the system level (e.g., when adaptations involve physical changes to the system that cannot be automated).

In particular, 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 itself (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, how can the system 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, affects the outcome of adaptation. In this paper, we investigate the problem of how the explicit modeling of human factors can provide a better insight into the trade-offs of involving humans in adaptation. In particular, we contribute a formal framework to reason about human involvement in self-adaptation, focusing on the role of human participants as actors (i.e., effectors) during the execution stage of adaptation. The backbone of the approach consists of two parts: (i) an extended version of a language to express adaptation models with elements that capture some of the main factors that affect human behavior and its interactions with the system, and (ii) a formalization of the extended adaptation models as stochastic multiplayer games (SMGs) that can be used to analyze human-system-environment interactions.

To explore these issues, we propose the extension of the Stitch language [12] employed in the Rainbow framework for self-adaptation [19] with elements inspired from opportunity-willingness-capability models employed in *cyber-human systems* [17] that capture major factors having an influence in human participant behavior. We illustrate our approach in the context of an adaptive industrial middleware that is used to monitor and manage highly populated networks of devices in renewable energy production plants.

This paper is organized as follows. In Section II, we present DCAS, the system that we employ to illustrate our approach. Section III introduces related work in the areas of self-adaptive systems, business process modeling, and cyber-human systems, outlining some of the major requirements for the approach. Section IV details the formal basis for mixed-initiative adaptation in the context of Stitch. In Section V we show how the formal modeling of human-in-the-loop adaptation in Rainbow can be leveraged through the use of probabilistic model checking of SMGs to reason about the combined behavior of human participants with a self-adaptive system. Finally, Section VI presents conclusions and directions for future work.

II. Motivating Scenario

The Data Acquisition and Control Service (DCAS) [5] is a middleware from Critical Software that provides a reusable infrastructure to manage the monitoring of highly populated networks of devices equipped with sensors. In particular, the middleware is designed to be seamlessly integrated with Critical's Energy Management System (csEMS)[1], which is a platform that provides asset management support for power producing companies based on renewable energy sources. The overall csEMS architecture aims at high scalability, flexibility and customization with management capabilities that enable the operation of control centers independently of the underlying application (*e.g.*, wind, solar, etc.).

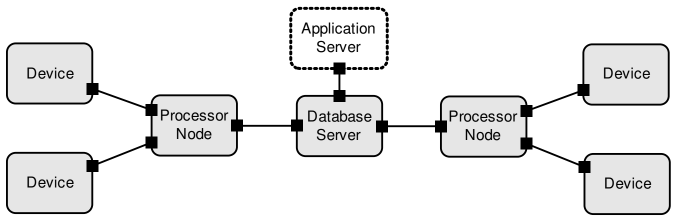

Fig. 1. Architecture of a DCAS-based system.

1) Overview of the Architecture: The basic building blocks in a DCAS-based system (Figure 1) are: [2]

- *Devices* are equipped with one or more sensors to obtain data from the application domain (e.g., from wind towers, solar panels, etc.). 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 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).

2) Adaptation Mechanisms: DCAS implements three adaptation mechanisms to maintain an acceptable level of performance while making an efficient use of computational resources:

- *Rescheduling* aims at avoiding performance degradation caused by devices that fail to respond in a timely manner when polled. It consists in decreasing the priority of the data streams associated with the failing devices, so that they are polled less often (thus reducing the average time that DRPPs remain blocked waiting for device data).
- *Scale up* aims at improving performance by exploiting as much as possible CPU and memory in active processor nodes by (de)activating DRPPs as required.
- *Scale out* aims at maintaining an acceptable performance level when new devices are connected to the network at runtime and all available resources in the set of active processor nodes are already being used. This is achieved 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

[1]http://solutions.criticalsoftware.com/products_services/csems/

[2]We herein consider a simplified version of the DCAS architecture. Further details about DCAS can be found in [5].

new processor node must be manually deployed, and devices re-attached across the different already active processor nodes, according to the particular situation. Specifically, the process followed by a human operator to perform scale out consists of the following steps:

1) determine which (possibly new) devices need to be attached to a processor node,
2) decide which of those devices can be attached to a currently active processor node, and which must be attached to a new one,
3) restart active processor nodes that have been assigned new devices, and
4) deploy and activate new processor node(s).

– As an *automated process* that can be executed by adaptation logic residing in an closed control loop.[3] 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 (pre-deployed) 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).

We illustrate our approach by focusing exclusively on the trade-offs of employing the two variants of scale out, since scale up and rescheduling are fully automated adaptations carried out locally on each processor node, according to the changing conditions of the devices that they are attached to.

III. 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* [16]) 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.

[3]In this paper, we refer to an automated version of scale out implemented on a Rainbow-based prototype of DCAS [4].

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 [27] or consider uncertainty when tuning the operation of the system (e.g., by dynamically adjusting parameters [3], [15]). 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 [19], [26] 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 [19]. However, 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 exclusively on fully automated adaptations, Dorn and Taylor [14] 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 [13], [28]. These languages target primarily service-oriented architectures and have limited or no support for other common architectural styles.

Eskins and Sanders [17] 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.

TABLE I
REQUIREMENTS SATISFIED BY RELATED APPROACHES.

Area	Approach	R1	R2	R3	R4	R5
Self-adaptive Systems	Sykes et al. [27]	✓				✓
	Calinescu et al. [3]		✓	✓		
	Epifani et al. [15]		✓	✓		
	Cheng et al. [19]	✓	✓	✓		✓
	Oreizy et al. [26]	✓				✓
	Dorn & Taylor [14]	✓			✓	✓
Business Process Modelling	BPMN [28]			✓	✓	
	WSBPEL4People [13]			✓	✓	
Cyber-Human Systems	Eskins & Sanders [17]		✓		✓	

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 I), in this paper we propose an approach that combines the strengths of the Rainbow framework for self-adaptation [19] and the OWC ontology described in [17]. 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 upon the target system, and is able to account for uncertainty in the selection 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 [7], 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.

IV. HUMAN-IN-THE-LOOP ADAPTATION IN RAINBOW

In this section, we first introduce the main concepts behind the Stitch language for self-adaptation, illustrating them with some examples in DCAS. Then, we present a candidate model for quantifying how human involvement in the execution of adaptation can affect system objectives. This model is inspired by the OWC ontology described in [17]. 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 actors.

A. Adaptation Model

Although many proposals that rely on closed-loop control exploit architectural models for adaptation [19], [23], [26], in this paper we use some of the high-level concepts in Stitch [12] as a reference framework to illustrate our approach. Stitch is the language employed by the Rainbow framework [19] 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 using the following high-level concepts:[4]

1) Tactic: is a primitive action that corresponds to a single step of adaptation. For instance, in DCAS we can specify pairs of tactics with opposing effects for (de)activating processor nodes, or scale up/down the number of DRPPs active in a given processor node. Listing 1 shows an example tactic for activating a processor node in DCAS. Line 7 specifies the applicability condition, which says that the tactic may be chosen if there is a processor node available not currently active (predicate defined in line 1). Lines 8-17 specify the action, which is to select a single processor node among those currently inactive (line 9), select a set of devices of size MAX_DEVS_PN (a constant that represents the maximum number of devices that can be attached to a processor node) among those that are currently not assigned to a processor node (line 10). Next, the selected devices are assigned to the processor node selected (lines 12-13), and finally the operation to enable the processor node is called (line 15). Line 18 states that the tactic succeeds only if all devices are attached to some processor node. Note that if the tactic starts with set.Size(UDevs) > M.MAX_DEVS_PN, the tactic will not be able to produce its intended effect because the selection of a subset of size M.MAX_DEVS_PN is done only once (line 10). This failure exit status for the tactic is useful as feedback indicating that more processor nodes may need to be activated.

```
 1  define boolean PNA=exists n:ProcessorNodeT in M.components |
        !n.isActive;
 2  define set ActivePNs={select n:ProcessorNodeT in M.components |
        !n.isActive};
 3  define set UDevs={select d:DeviceT in M.components | d.location=−1};
 4  define boolean unprocessedDevices=set.Size(UDevs)>0;
 5  ...
 6  tactic activatePN () {
 7    condition {PNA}
 8    action {
 9      set pn = set.RandomSubset (ActivePNs,1);
10      set devs=set.RandomSubset (UDevs, M.MAX_DEVS_PN);
11      for (ProcessorNodeT n : pn) {
12          for (DeviceT d : devs) {
13              M.assignDeviceToPN(d, n.id);
14          }
15           M.enablePN(n, true);
16      }
17    }
18    effect {!unprocessedDevices;}
19  }
```

Listing 1. Tactic for activating a processor node in DCAS.

Tactics have an associated cost/benefit impact on the different dimensions of concern in the system. Table II shows

[4]We use a simplified version of Stitch [12] to illustrate the main ideas in this paper.

the different tactics in DCAS, and their impacts on quality dimensions:[5] activatePN is an automated tactic that activates a previously deployed processor node, reducing response time and increasing operation cost, whereas addPN is carried out by a human actor and has also a beneficial impact on performance, and detrimental in cost. Although the impact on cost is the same as in activatePN, the impact on performance will typically be higher, since a human operator is less constrained than the automated approach to exploit the available processor nodes by re-attaching devices properly. However, the downside of tactic addPN is that it typically has a much higher latency (i.e., the time it takes since a tactic is triggered, until its effects can be observed) than activatePN. Note that, to obtain both the latency and 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. Although in this paper we consider fixed cost/benefit impacts for illustration purposes, Stitch also supports the specification of sophisticated impact models that are context-sensitive, and can capture probabilistic aspects in the outcome of tactic executions [6].

TABLE II
COST/BENEFIT FOR DCAS TACTICS.

Tactic	Performance (P)		Cost (C)	
	Δ Requests Per Second (rps)	ΔU_P	Δ Operating Cost (usd/hr)	ΔU_C
activatePN	+80	$\uparrow$	+1.0	$\downarrow$
addPN	+150	$\uparrow\uparrow$	+1.0	$\downarrow$

2) 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 ScaleOut
2 [unprocessedDevices] {
3     t0a: (maxAssignedDevices) −> : activatePN()@[10000] {
4         t0aa: (success) −>done;
5         t0ab: (fail) −> TNULL;
6     }
7     t0b: (fail) −> TNULL;
8 }
```

Listing 2. Strategy for activating a processor node automatically.

Listing 2 shows the Stitch code for a simple scale out adaptation strategy in DCAS: line 2 specifies the applicability condition that needs to be satisfied for the strategy to be eligible for execution (in this case, predicate unprocessedDevices indicates that there are new devices in the network that need to be attached to a processor node). In the body of the strategy, node t0a (line 3) executes tactic activatePN if the guard maxAssignedDevices evaluates to true (i.e., if the set of active processor nodes are already attached to the maximum number of devices they support). To account for the delay in observing the outcome of tactic execution in the system (settling time), t0a specifies a time window of 10 seconds, after which, if the tactic's intended effect (as defined in the tactic script – Listing 1, line 18) is observed, successful tactic completion (keyword success, line 4) leads to the end strategy execution in normal conditions through node t0aa (keyword done). Otherwise, if the intended tactic effect is not observed after the delay window expires (keyword fail, line 5), the strategy exits with an error status via node t0ab. An additional condition-action-delay node (t0b, line 7) also exits the strategy with an error status if the guard of node t0a is false, and tactic activatePN cannot be executed in the first place.

3) Utility Profile: to enable the selection of strategies at runtime, we assume that adaptation 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. Table III summarizes an example of utility functions for DCAS. Function U_P maps high levels of processed requests per second inserted in the database (rps) to high utility by dividing it by the maximum level of achievable rps in the system rps^{max}, which is computed according to the number of devices in the system and the data polling rates configured in their device profiles.

In contrast, function U_C maps higher costs (derived from the number of active processor nodes – pn) to lower utility values. Cost utility becomes 0 when pn reaches the maximum number of available nodes pn^{max}. Utility preferences capture business preferences over the quality dimensions, assigning a weight to each one of them (e.g., weights $w_{U_P} = 0.8$ and $w_{U_C} = 0.2$ indicate that performance is the main concern in the system).

TABLE III
UTILITY FUNCTIONS FOR DCAS.

Performance (P)	Cost (C)
$U_P(rps) = \frac{rps}{rps^{max}}$	$U_C(pn) = 1 - \frac{pn}{pn^{max}}$

B. 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 [17]. 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). In particular, attributes of human actor types can be categorized into:

1) Opportunity: captures the applicability conditions of the adaptation tactics that can be executed by human actors upon

[5]The table also provides an intuition of how the different tactics affect the utility for every particular dimension (the number of upward or downward arrows is directly proportional to the magnitude of utility increments and decrements, respectively).

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.

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 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 $pr_W : S \to [0, 1]$.

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 $CE^{\mathsf{addPN}} = \{T\}$, 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 $pr_C : T \to [0, 1]$.

C. 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.

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 processor nodes to deploy?). In general, applicability conditions of these tactics will be a combination of both. In Listing 3, we can observe how the condition block of tactic addPN (line 5) 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 (processor node available – PNA, defined in line 2).

The action block of these tactics can execute automated operations, as in the case of tactic activatePN (Listing 1), but also notify human actors to perform a task. The action block of of tactic addPN (Listing 3, lines 6-8) first selects an available processor node (line 6), and then an available operator on location (line 7). Finally, it notifies the selected operator that she has to activate the previously selected processor node via a text message communicating its id.

```
 1 define boolean ONLNB=exists o:operatorT in M.participants | o.onLocation
        && !o.busy;
 2 define boolean PNA=exists n:ProcessorNodeT in M.components |
        !n.isActive;
 3 ...
 4 tactic addPN(){
 5   condition {ONLNB && PNA;}
 6   action {pn=Set.RandomSubSet(ActivePNs,1);
 7           ao=Set.RandomSubSet({select o:operatorT in M.participants
                 | o.onLocation && !o.busy},1);
 8           notify(ao, "Deploy processor node: %s", pn.id);}
 9   effect {!unprocessedDevices;}
10 }
```

Listing 3. Tactic for adding a processor node via human operator.

2) Strategies: Fully automated, as well as tactics involving humans can be combined to achieve better outcomes in adaptation strategies. Listing 4 shows strategy ScaleOutOp for scaling out the system, first by notifying an operator (via tactic addPN, line 3) to manually deploy and activate a processor node. If the assigned time window of 50 seconds expires and the intended effect of the tactic (!unprocessedDevices) is not observed, the strategy executes the automated activatePN tactic as a fallback mechanism (line 4).

```
 1 strategy ScaleOutOp
 2 [unprocessedDevices] {
 3    t0 : (maxAssignedDevices) −> addPN()@[50000] { // add PN, wait 50s
 4       t0a: (fail) −> : activatePN()@[10000] { // If failed, activate PN
 5          t0aa: (success) −>done;
 6          t0ab: (fail) −> TNULL;
 7       }
 8       t0b: (fail) −> TNULL;
 9    }
10 }
```

Listing 4. Strategy combining automated/manual tactics for activating a processor node.

V. 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 in Section IV 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 automatic 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 DCAS SMG model implemented in the probabilistic model-checker PRISM-games [9], as well as the analysis and results that we obtained for human-in-the-loop adaptation in the context of DCAS scale out.

A. Model Checking Stochastic Multiplayer Games

Automatic verification techniques for probabilistic systems have been successfully applied in a variety of application domains that range from power management or wireless communication protocols, to biological systems. 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, 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 [24] or communication protocols [20].

Our approach to analyzing human involvement in adaptation builds upon a recent technique for modeling and analyzing SMGs [8]. 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 [10]. PATL is itself an extension of ATL [1], 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[6] 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 $\mathsf{P}_{\bowtie \mathsf{q}}$ and path formulae from PCTL [2]. Moreover, rPATL includes a generalization of the reward operator $\mathsf{R}^{\mathsf{r}}_{\bowtie \mathsf{x}}$ from [18] to reason about goals related to rewards. An extended version of the rPATL reward operator $\langle\langle C \rangle\rangle \mathsf{R}^{\mathsf{r}}_{\mathsf{max}=?}[\mathsf{F}^{\star}\ \phi]$ [7] enables the quantification of the maximum accrued reward r along paths that lead to states satisfying state formula ϕ that can be guaranteed by players in coalition C, independently of the strategies followed by the rest of players. An example of typical usage combining the coalition and reward maximization operators is $\langle\langle \mathsf{sys} \rangle\rangle \mathsf{R}^{\mathsf{utility}}_{\mathsf{max}=?}[\mathsf{F}^{\mathsf{c}}\ \mathsf{end}]$, meaning "value of the maximum utility reward accumulated along paths leading to an end state that a player sys can guarantee, regardless of the strategies of other players."

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.[8]

B. Formal Model

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

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 (target system), as well as scaleOutOp and scaleOut (adaptation strategies for manual and automatic scale out, respectively). Moreover, the system player controls the synchronization of actions between scale out adaptation strategies and the target system, that represent the addPN and activatePN adaptation tactics. Global variable turn (line 4) is used to explicitly encode alternating turns between the system and environment players.

```
1  player env environment endplayer
2  player sys ha_system, [addPN], [activatePN], scaleOutOp, scaleOut
        endplayer
3  const ENVT=0, SYST=1;
4  global turn:[ENVT..SYST] init SYST;
```

Listing 5. Player definition for the DCAS scale out SMG.

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

[6]The term *strategy* employed in the context of SMGs refers to a *game strategy* (referred to also as *policy* or *adversary*) as defined in [8], and should not be confused with Stitch adaptation strategies.

[7]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.

[8]See [8] for more details on SMG strategy synthesis.

```
1  const MAX_TIME; // Execution 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.

In Listing 6, variable t (line 3) keeps track of execution time (the time frame for the system's execution is determined by [0, MAX_TIME][9]). During its turn, the environment checks that the end of the time frame for the execution has not been reached yet, and if that is the case, it increments the value of t one unit, yielding the turn to the system player (line 4)[10].

3) Human Model: Listing 7 shows the encoding of the OWC elements corresponding to an operator in the DCAS system. Opportunity elements (line 2) indicate whether the operator is on location and/or busy. These predicates are used to guard the execution of tactic addPN in the model (Listing 8, line 9). The willingness function of the operator (line 6) is inversely proportional to the stress level of the operator, declared in line 5. The capability function (line 9) penalizes training levels below 0.4, whereas it rewards levels above 0.8.

```
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 > 0.8 ? 1 : (op_trainingLevel<0.4?
        op_trainingLevel/3 : op_trainingLevel);
10 // Combined WC probability for tactic addPN
11 formula addPN_wc_prob = op_f_c ∗ op_f_w;
```

Listing 7. Human actor model encoding for a DCAS operator.

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 2-6 illustrate how the different variables are initialized:

- rps and pn encode the performance and number of active processor nodes in the DCAS system, respectively.
- addpn_fail is a flag that determines if the addition of a processor node by a human actor failed.
- cnt_addPN and cnt_activatePN are counters used to keep track of the latency of tactics addPN and activatePN, respectively.

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 addPN to illustrate how tactic execution is modeled:

- Tactic trigger (line 9). Triggers tactic execution 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 (cnt_addPN'=1).
- Tactic latency counter update (line 12). 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 15). When the tactic's latency counter expires, the command can either: (i) update variables rps and pn according to a successful activation of a processor node with probability addPN_wc_prob (determined by the willingness and capability elements defined in Listing 7, line 11), or (ii) fail to activate the node with probability 1-addPN_wc_prob, flagging the failure on variable addPN_fail. In both cases, the latency counter is reset, and the busy status of the operator is set to false.

The encoding used for the activatePN tactic (lines 10,13,18) 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 20) 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).

5) Adaptation logic: Modules scaleOutOp and scaleOut 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 9-10).

Module scaleoutOp (Listing 9) models the variant of the scale out mechanism that makes use of a human actor. The command on line 3 encodes the triggering of tactic addPN [11], which sets the value of the timestamp variable addPN_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 activatePN (line 4) to determine whether the

[9]Constant values not defined in the model are provided as command-line input parameters to the tool.

[10]We illustrate our approach to modeling the SMG using the syntax of the PRISM language [25] for Markov Decision Processes (MDPs), which are encoded as commands:

$$[action]\ guard\ {-}{>}\ p_1 : u_1 + \ldots + p_n : u_n$$

Where $guard$ is a predicate over the model variables. 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 probably, including a single update of unspecified probability) introduce local nondeterminism.

[11]We abstract away predicates unprocessedDevices and maxAssignedDevices (Listing 4, lines 2,3), which we assume to be true in the scale out scenarios encoded in our model.

```
1  module ha_system
2  rps : [RPS_MIN..RPS_MAX] init INIT_RPS;
3  pn : [PN_MIN..PN_MAX] init INIT_PN;
4  addpn_fail: bool init false;
5  cnt_addPN :[0..addPN_settling] init 0;
6  cnt_activatePN :[0..activatePN_settling] init 0;
7
8  // Tactic triggers
9  [addPN] (turn=SYST) & (op_onLocation) & (!op_busy) & (pn<PN_MAX) &
          (cnt_addPN=0) -> (cnt_addPN'=1) & (op_busy'=true);
10 [activatePN] (turn=SYST) & (pn<PN_MAX) & (cnt_activatePN=0) ->
          (cnt_activatePN'=1) & (turn'=ENVT);
11 // Tactic latency counter update
12 [] (turn=SYST) & (cnt_addPN>0) & (cnt_addPN<addPN_latency) ->
          (cnt_addPN'=cnt_addPN+1) & (turn'=ENVT);
13 [] (turn=SYST) & (cnt_activatePN>0) &
          (cnt_activatePN<activatePN_latency) ->
          (cnt_activatePN'=cnt_activatePN+1) & (turn'=ENVT);
14 // Tactic completion (after latency period expires)
15 [] (turn=SYST) & (cnt_addPN=addPN_latency) ->
16           addPN_wc_prob : (cnt_addPN'=0) & (rps'=addpn_f_p) &
                    (pn'=f_apn) & (op_busy'=false) & (turn'=ENVT)
17                 + 1-addPN_wc_prob : (cnt_addPN'=0) & (addpn_fail'=true)
                          & (op_busy'=false) & (turn'=ENVT) ;
18 [] (turn=SYST) & (cnt_activatePN=activatePN_latency) ->
          (rps'=activatepn_f_p) & (cnt_activatePN'=cnt_activatePN+1) &
          (pn'=f_apn) & (cnt_activatePN'=0) & (turn'=ENVT);
19 // Do nothing
20 [] (turn=SYST) & (cnt_addPN=0) & (cnt_activatePN=0) -> (turn'=ENVT);
21 endmodule
```

Listing 8. Target system extended with human actors module.

settling time for observation of the previous tactic's effect has already expired. If this is the case, and the activation of the new processor node by the human operator is not successful (addpn_fail), the command executes, triggering the activatePN tactic, consistently with the Stitch code in listing 4, line 4.

```
1  module scaleOutOp
2  addPN_trigger_t:[0..MAX_TIME] init 0;
3  [addPN] (turn=SYST) -> (addPN_trigger_t'=t);
4  [activatePN] (turn=SYST) & (t=addPN_trigger_t+addPN_settling) &
          (addpn_fail) -> true;
5  endmodule
6
7  module scaleOut
8  [activatePN] (turn=SYST) -> true;
9  endmodule
```

Listing 9. scaleOutOp and scaleOut adaptation strategy modules.

Module scaleOut (Listing 9, line 7) models the automatic scale out mechanism with a single command that triggers the tactic's execution on the target system.

6) Utility profile: Utility functions and preferences are encoded using formulas and reward structures that enable the quantification of instantaneous utility (Listing 10). In particular, formulas quantify utility for every dimension of concern (lines 1-2) according to utility functions described in Table III, and a reward structure rGU (line 4) weighs them against each other by using the utility preferences specified as weights for performance and cost (line 3).

```
1  formula uP=(rps/RPS_MAX); // Performance
2  formula uC=1-(pn/PN_MAX); // Cost
3  const double wP = 0.8, wC = 0.2; // Utility preferences
4  rewards "rGU" turn=SYST : wP * uP + wC * uC; // Instantaneous utility
5  endrewards
```

Listing 10. Utility profile for DCAS SMG.

C. 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.

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

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

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 2(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]. Tactic cost/benefit values and the utility profile employed are those described in Section IV-A, whereas the latency values employed for tactics activatePN and addPN are 5 and 30 seconds, respectively. Time delays to observe the effect of tactic executions in strategies ScaleOut and ScaleOutOp are those indicated in the Stitch code shown in Listings 2 and 4, respectively.

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 at a value of 112.75, since it is not affected by willingness or capability factors.

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:

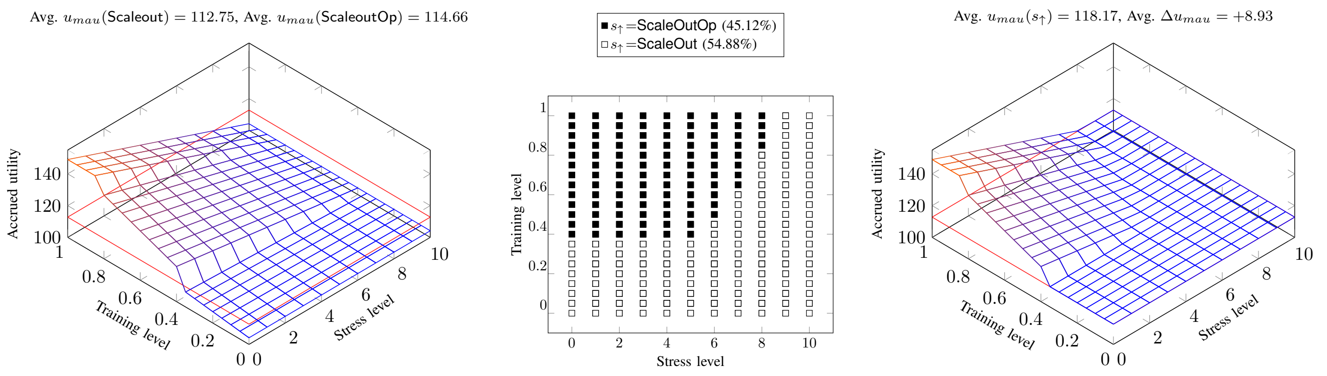

Fig. 2. Experimental results: (a) ScaleOut *vs* ScaleOutOp strategy utility (left), (b) strategy selection (center), and (c) combined utility (right).

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

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

Figure 2(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 op_f_c in Listing 7, which highly penalizes poorly trained operators – op_trainingLevel ≤ 0.4).

Figure 2(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[12] in the combined solution is +8.93, corresponding to a percentual improvement of 7.94% over the automatic scale out approach, and 7.81% over the manual one.

VI. Conclusion

In this paper, we have described an approach that employs formal reasoning about human involvement in the adaptation loop of self-adaptive systems to provide a systematic approach to human-in-the-loop adaptation. 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). Moreover, we have 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 scale out adaptation in DCAS, an industrial middleware for monitoring large sensor networks in renewable energy power plants. Our results showed that our approach can help 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 in a context-sensitive manner.

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 [17]. 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. Finally, we plan on performing live experiments with human actors for validating our results.

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-0002084).

[12]We employ the delta in accrued utility as an indicator of the improvement in using a combined solution with respect to a manual/automated adaptation, defined as $\Delta u_{mau} \triangleq |u_{mau}(s_\uparrow) - u_{mau}(\arg\min_{s \in \mathcal{S}} u_{mau}(s))|$, with $\mathcal{S} = \{\mathsf{ScaleOut}, \mathsf{ScaleOutOp}\}$.

References

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

[2] A. Bianco and L. de Alfaro. Model checking of probabalistic and nondeterministic systems. In P. S. Thiagarajan, editor, *Foundations of Software Technology and Theoretical Computer Science, 15th Conference, Bangalore, India, December 18-20, 1995, Proceedings*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513. Springer, 1995.

[3] R. Calinescu and M. Z. Kwiatkowska. Using quantitative analysis to implement autonomic IT systems. In J. M. Atlee and P. Inverardi, editors, *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings*, pages 100–110. IEEE, 2009.

[4] J. Cámara, P. Correia, R. de Lemos, D. Garlan, P. Gomes, B. Schmerl, and R. Ventura. Incorporating architecture-based self-adaptation into an adaptive industrial software system. Submitted for publication. Available at: http://acme.able.cs.cmu.edu/pubs/show.php?id=431.

[5] 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.

[6] J. Cámara, A. Lopes, D. Garlan, and 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*, volume 8997 of *Lecture Notes in Computer Science*, pages 89–107. Springer, 2014.

[7] 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.

[8] 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.

[9] T. Chen, V. Forejt, M. Z. Kwiatkowska, D. Parker, and A. Simaitis. Prism-games: A model checker for stochastic multi-player games. In N. Piterman and S. A. Smolka, editors, *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*, volume 7795 of *Lecture Notes in Computer Science*, pages 185–191. Springer, 2013.

[10] T. Chen and J. Lu. Probabilistic alternating-time temporal logic and model checking algorithm. In J. Lei, editor, *Fourth International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2007, 24-27 August 2007, Haikou, Hainan, China, Proceedings, Volume 2*, pages 35–39. IEEE Computer Society, 2007.

[11] 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.

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

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

[14] C. Dorn and R. N. Taylor. Coupling software architecture and human architecture for collaboration-aware system adaptation. 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 53–62. IEEE / ACM, 2013.

[15] 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.

[16] 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.

[17] 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.

[18] V. Forejt, M. Z. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In M. Bernardo and V. Issarny, editors, *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*, volume 6659 of *Lecture Notes in Computer Science*, pages 53–113. Springer, 2011.

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

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

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

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

[23] J. Kramer and J. Magee. Self-managed systems: an architectural challenge. In L. C. Briand and A. L. Wolf, editors, *International Conference on Software Engineering, ISCE 2007, Workshop on the Future of Software Engineering, FOSE 2007, May 23-25, 2007, Minneapolis, MN, USA*, pages 259–268, 2007.

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

[25] M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011.

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

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

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