Uncertainty Reduction in Self-Adaptive Systems

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

Javier Cámara
jcmoreno@cs.cmu.edu
Carnegie Mellon University
School of Computer Science
Pittsburgh, PA, USA

David Garlan
garlan@cs.cmu.edu
Carnegie Mellon University
School of Computer Science
Pittsburgh, PA, USA

Mark Klein
mk@sei.cmu.edu
Carnegie Mellon University
Software Engineering Institute
Pittsburgh, PA, USA

ABSTRACT

Self-adaptive systems depend on models of themselves and their environment to decide whether and how to adapt, but these models are often affected by uncertainty. While current adaptation decision approaches are able to model and reason about this uncertainty, they do not consider ways to *reduce* it. This presents an opportunity for improving decision-making in self-adaptive systems, because reducing uncertainty results in a better characterization of the current and future states of the system and the environment (at some cost), which in turn supports making better adaptation decisions. We propose *uncertainty reduction* as the natural next step in uncertainty management in the field of self-adaptive systems. This requires both an approach to decide *when* to reduce uncertainty, and a catalog of tactics to reduce different kinds of uncertainty. We present an example of such a decision, examples of uncertainty reduction tactics, and describe how uncertainty reduction requires changes to the different activities in the typical self-adaptation loop.

ACM Reference Format:
Gabriel A. Moreno, Javier Cámara, David Garlan, and Mark Klein. 2018. Uncertainty Reduction in Self-Adaptive Systems. In *SEAMS '18: SEAMS '18: 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems , May 28–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3194133.3194144

1 INTRODUCTION

The software-intensive systems that our society relies on are increasingly expected to satisfy their functional and extra-functional requirements under changing conditions in the systems and in their environments, including fluctuations in user demand and resource availability, component failures, and the presence of cyber adversaries. Given the scale of modern systems and the fast pace at which run-time conditions change, it is not viable to rely mainly on humans to reconfigure systems to maintain optimal performance. In safety-critical domains, such as unmanned vehicles operating in remote areas and unmanned spacecrafts, reliance on humans is not even an option. Self-adaptation is an approach to deal with this problem by engineering a system with the ability to monitor its own state and the state of its environment, and to autonomously change its structure and behavior to operate as well as possible with respect to a defined goal in the presence of run-time changing conditions.

A self-adaptive system cannot make arbitrary changes to itself, but instead uses a repertoire of *adaptation tactics*—action primitives that change the system leaving it in a consistent state [13] (e.g., instantiating a new server, lowering the fidelity of a computation, or turning a sensor off). In order to decide whether to adapt and how to do it (i.e., which adaptation tactics to use), self-adaptive systems depend on models of themselves and their environment. These models are often affected by uncertainty that stems from different factors [14, 35]. Two important classes of uncertainty are those due to variability or and lack of knowledge [14]. The former, also known as *aleatory uncertainty*, is due to randomness, whereas the latter, *epistemic uncertainty*, is due to the lack of knowledge about the state of the system (or environment) and not due to variability.

As previous research has shown, explicitly taking uncertainty into account improves the effectiveness of self-adaptation [5, 11, 14]. For example, consider a system that must keep a parameter below a threshold. If it uses a simple point estimate of that parameter (e.g., the mean of the observations) to decide whether to adapt, and that point estimate is below the threshold, the system would decide not to adapt. On the other hand, using a confidence interval to characterize the uncertainty of that estimate could help the system determine that it is possible that the parameter is above the threshold even if the point estimate is not, thus potentially prompting an adaptation.

While current adaptation decision approaches are able to model and reason about the uncertainty associated with a system and its environment, they do not consider ways to *reduce* uncertainty. In some cases, that is simply not possible because the uncertainty is irreducible (e.g., due to an action controlled by an external entity for which there is no model), but in other cases, the uncertainty is reducible because it is about things that are knowable but are unknown at a given time (e.g., a ping to a server can reduce the

SEAMS '18, May 28–29, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5715-9/18/05…$15.00
https://doi.org/10.1145/3194133.3194144

uncertainty of whether it is alive) [14]. This presents an opportunity for improving decision-making in self-adaptive systems, because uncertainty reduction results in a better characterization of the current and future states of the system and the environment, which in turn supports making better adaptation decisions. However, reducing uncertainty has a cost due, for example, to resources consumed or time taken. This presents a trade-off between the benefit and the cost of uncertainty reduction that must be considered when evaluating the impact of uncertainty reduction.

In this paper, we propose *uncertainty reduction* as the natural next step in uncertainty management in self-adaptive systems. This requires both an approach to decide *when* to reduce uncertainty, and specific techniques to reduce different kinds of uncertainty. For the latter, we propose using *uncertainty reduction tactics*, such as sending a request to a web service that has not been recently used to get information about its availability and response time [19], which reduces the uncertainty that arises from lack of knowledge; and getting more observations of a parameter to obtain a narrower confidence interval, reducing uncertainty due to variability. Although a variety of uncertainty reduction tactics like these exist and have been used in practice, to date their use has been ad hoc, and has not taken into consideration the tradeoff—in terms of costs and benefits—in using them as is done with regular adaptation tactics. For this reason, a principled decision-making approach to decide when to use uncertainty reduction tactics is needed.

With a simple motivating example, we present the idea of uncertainty reduction and how a decision-making approach can choose when to use an uncertainty reduction tactic. Besides new decision-making approaches, uncertainty reduction also requires capturing the information needed to make such decisions, executing uncertainty reduction tactics, and capturing their output. We show how these activities can be mapped to the MAPE-K loop [22], and present changes in the interactions between the different parts of the loop that would be needed. In addition, we describe some uncertainty reduction tactics with examples of how they can be used.

Although our initial results are promising, to enable developers of self-adaptive systems to fully exploit existing and emerging techniques for uncertainty reduction, two things are needed: *(i)* a catalog of uncertainty reduction tactics, together with descriptions (preferably formal) of their costs and benefits; and *(ii)* tractable reasoning mechanisms that allow a self-adaptive system to seamlessly integrate uncertainty reduction with other adaptation tactics. Achieving this goal presents challenges, which we also discuss.

The rest of this paper is organized as follows. Section 2 presents a motivating example and shows how uncertainty reduction works. Section 3 describes how the different activities typically present in a self-adaptation loop have to be extended to realize uncertainty reduction. Examples of tactics and the kind of uncertainty they reduce are presented in Section 4. Section 5 concludes the paper with a description of the research challenges that realizing uncertainty reduction poses.

2 MOTIVATING EXAMPLE

Consider a corporate system comprised of two servers: server *A* is a web server, and server *B* is a database. For this example, let us focus on self-adaptation for protecting against a cyber-attack (a.k.a., self-protection [44]). An intrusion detection system (IDS) has just raised an alert indicating that there is a 30% chance of server *A* being compromised, and since nothing suspicious has been detected on the other server yet, the IDS assumes it is clean. This current state is represented by the leftmost circle in Figure 1.

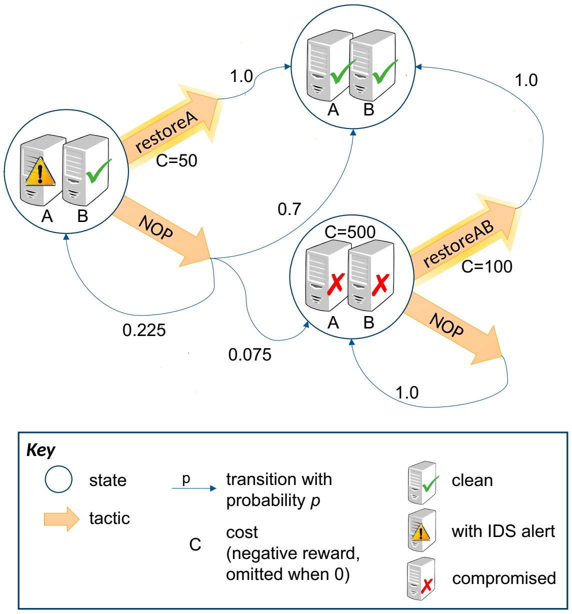

Figure 1: Decision without uncertainty reduction.

The system has to make a decision between using tactic *restoreA* to restore the contents of the server *A* to a non-compromised version from a backup, or do nothing, which we refer to as the *NOP* tactic. These two alternatives are depicted as thick arrows going out of the current state. The tactic *restoreA* has a cost $C = 50$,[1] but when used, it results in server *A* being clean with certainty (as depicted by the single transition with probability 1.0). Tactic *NOP*, on the contrary, has no cost, but has three possible outcomes in our model. Since the IDS estimates that the server *A* is compromised with probability 0.3, server *A* is actually clean with probability 0.7. If server *A* is compromised, we assume that there is a small probability (0.075) that the attacker—who is part of the environment for this system—will move on to compromise server *B*. In that case, there is a big cost incurred ($C = 500$) due to the sensitivity of the data in that server. On the other hand, if the attacker does not move on to server *B*, the system remains in the same state.

In situations like this, making decisions solely on the immediate outcome of the possible action the system can take, can result in suboptimal adaptation. In this example, a myopic decision would consider that the cost of using tactic *restoreA* is higher than the expected cost of doing nothing ($1.0 \cdot 50 > 0.075 \cdot 500 = 37.5$). Instead, a better decision can be made by considering what would happen beyond the current decision, including the actions the system and the attacker could take in the future. To that end, the possible tactics

[1] Tactics can incur costs due to resources or time consumed, for example. In this case, the cost could be the revenue lost by the system being down for the duration of the restore operation.

Table 1: Tactic choice without uncertainty reduction as a function of the probability of server *A* being compromised.

$p = P(A \text{ compromised})$	tactic
$p \leq 0.267$	NOP
$p > 0.267$	restoreA

in the state in which both servers are compromised represent the system actions in that state. If the system does not do anything (i.e., tactic *NOP*), the system keeps incurring a high cost due to the additional time the attacker has to exfiltrate sensitive data.[2] Tactic *restoreAB*, has a cost of $C = 100$, but evicts the attacker from the two servers.

A model like the one shown in Figure 1 is a Markov decision process (MDP), which can be used to make sequential decisions under uncertainty [33]. A policy for an MDP is a mapping from state to action[3] indicating which action must be taken in each state. An optimal policy for our example is such that when followed, it minimizes the expected cost. The optimal policy for an MDP can be generated using, for example, the PRISM probabilistic model checker [25]. Table 1 shows the optimal decision (i.e., tactic to use in the current state) depending on the probability of server A being compromised. For our example with $p = 0.30$, the optimal policy indicates that the system should use tactic *restoreA* to minimize the expected cost, which is 50 under this policy. Given the uncertainty about the state of the system, and the high cost of not restoring server A if it happens to be compromised, the optimal decision in this case is to execute tactic *restoreA* at a cost *even though it is more likely that server A is not compromised.* This raises the question of whether more effective adaptation decisions (i.e., with reduced expected cost) could be made if the system could have a better characterization of the state of server A at the expense of incurring a cost to do so. This is analyzed next.

To show how uncertainty reduction can improve the effectiveness of self-adaptation, let us revisit the example, but now with the addition of an uncertainty reduction tactic. Tactic *scanA* performs a malware scan in server A and reports whether it is infected or not. This tactic has a cost (e.g., due to the performance impact it causes, but a far less impact than restoring the server), so it would not be effective to use it at all times. In order to decide when to use it, we have to include it in the decision process, modeling not only its cost, but also its impact. Figure 2 shows the MDP with this tactic added. In this case, the outcome of this tactic is modeled using the intrusion probability reported by the IDS. That is, we assume that the tactic will report that the server is clean with 70% probability, or compromised with 30% probability. The latter case is modeled with a new state which represents the knowledge of the state of server A with reduced uncertainty.

As was done before, the decision is made by computing the optimal policy that minimizes the expected cost. In this case, it indicates that the optimal decision is to execute the tactic *scanA*, achieving an overall expected cost of 40, a 20% reduction compared to not having this tactic available. Although this may seem an obvious answer, it is not obvious that having an available uncertainty reduction tactic does not imply that it is always optimal to use it, as this depends on the rest of the model. Table 2 shows how the decision changes depending on the probability of server A being compromised as reported initially by the IDS. If this probability is more than 0.5, the system directly executes the *restoreA* tactic without attempting to reduce the uncertainty first. Intuitively, this is consistent with the fact that it may not be worth incurring the extra cost of reducing uncertainty, since the server is most likely compromised anyway. When the probability of server A being compromised is no more than 0.198, the system does not do anything, even though uncertainty reduction is an option. As it can be gleaned from these results, a decision like this can take into account the tradeoff between the cost and benefits of reducing uncertainty.

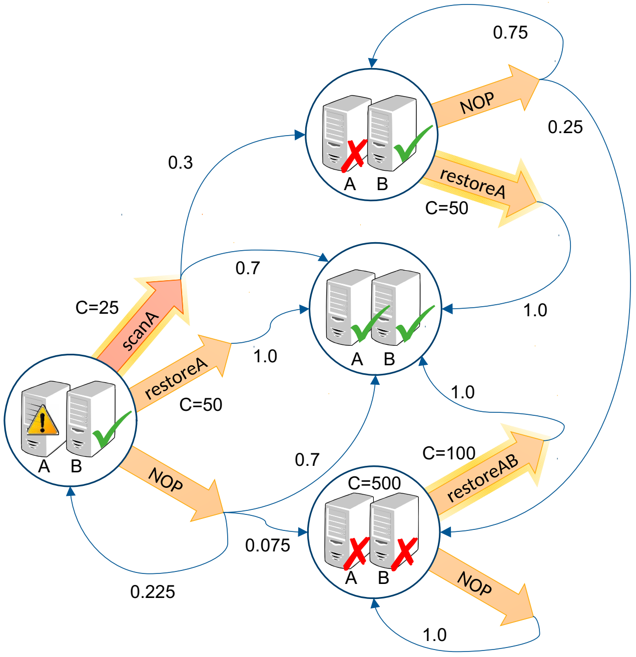

Figure 2: Decision with uncertainty reduction.

Table 2: Tactic choice with uncertainty reduction as a function of the probability of server *A* being compromised.

$p = P(A \text{ compromised})$	Tactic
$p \leq 0.198$	NOP
$0.198 < p \leq 0.5$	scanA
$p > 0.5$	restoreA

In this paper we argue that this example is one instance of a broad class of problems that can benefit from a systematic approach to uncertainty reduction. In the following sections, we describe what such an approach would require, the challenges it poses, and present additional examples.

[2] This is modeled as a self-transition, incurring a cost of $C = 500$ every time step in which *NOP* is taken and the system remains in the same state.

[3] This is a Markovian deterministic policy; however, there are other kinds of policies (see [34]).

3 UNCERTAINTY REDUCTION

A principled approach to uncertainty reduction requires not only deciding when to reduce uncertainty and how, but also capturing the information necessary to make that decision, executing the uncertainty reduction tactics, and capturing the information they produce. These activities and the information they use can be mapped to the MAPE-K loop shown in Figure 3 [22]. The following sections describe the extensions that would be needed to support uncertainty reduction.

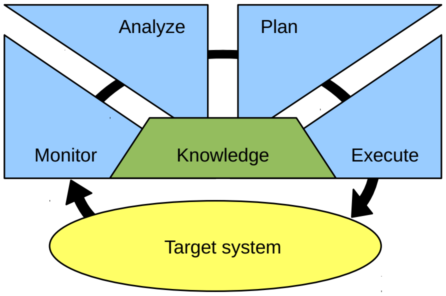

Figure 3: MAPE-K loop.

Monitor. This activity has to capture the information necessary to make uncertainty reduction decisions. In the example presented in Section 2, this activity would monitor the probability of server *A* being compromised as reported by the IDS, but in general, it would need to collect the probability distribution over the possible states. For other kinds of uncertainty reduction (e.g., to reduce a confidence interval), it may need to gather meta-information about a particular parameter, such as the number of observations taken of that parameter.

In approaches based on MAPE-K, such as Rainbow [17] and Hogna [2], *monitor* is the only activity that modifies the *knowledge*. That is, when an adaptation tactic is executed to modify the system, that change is not reflected in the *knowledge* until it is detected by *monitor*, providing robustness against tactic execution failure. Preserving the same rules for uncertainty reduction tactics would require that the *monitor* activity captures the output of that tactic (e.g., the output of the malware scan). Another alternative would be to allow uncertainty reduction tactics to directly record their output in the *knowledge*, since a failure of such a tactic would result the *knowledge* not being updated, as if the tactic had not been executed.[4] This would remove the indirection and the associated development effort needed to capture the output of the tactic via *monitor*.

Finally, some uncertainty reduction tactics may change the *monitor* activity instead of the system. This is the case when the sampling rate of a parameter must be increased, or when a parameter that was not being monitored must start being monitored. This requires a modification to the MAPE-K loop so that the *execute* activity can effect changes on the *monitor* (see *Execute* below for more details).

Analyze. The analysis has to determine whether the system would benefit from reducing uncertainty. This decision is different than the traditional responsibility of the *analyze* activity, which decides whether the system needs to adapt. Adaptation can be (and often is [24, 37]) reactive—based on the current state of the system and environment. However, deciding whether to reduce uncertainty requires assessing the impact of doing so on subsequent adaptation decisions, something that reactive approaches cannot do. For this reason, approaches that make adaptation decisions with a look-ahead horizon, such as the work of Gerasimou et al. [18] and Moreno et al. [31], may be more suitable as starting points for developing this kind of analysis.

It would also be possible to combine *analyze* with *plan*, as is done in the proactive self-adaptation approach PLA, which does not separate analysis and planning activities in its adaptation loop because deciding how to adapt requires evaluating the impact of the possible adaptations on the subsequent decisions [30, 31]. In fact, this combination was done in the example of the previous section, when we analyzed the model of Figure 2. The computed optimal policy determined that uncertainty reduction was beneficial *and* indicated the tactic we had to use.[5]

Plan. If the *analyze* activity determines that uncertainty reduction is needed, the *plan* activity must select one or more uncertainty reduction tactics to execute based on some criteria such as minimizing the expected cost (as in our example) or maximizing utility. The challenge here is that uncertainty reduction by itself does not have any benefit, so such assessment has to be made taking into account how the different applicable uncertainty reduction tactics will improve subsequent decisions. In the example shown in Figure 2, this was done leveraging an MDP to model a sequential decision problem. However, this is a simple example and challenges still remain to incorporate uncertainty reduction to a broad class of problems. First, in this case it was possible to explicitly represent the only two possible outcomes of the *scanA* tactic, but it is not clear how the impact of other tactics, such as one that increases the sampling rate of a parameter, should be represented. Second, in the example we limited the applicability of uncertainty reduction to a single state. Adding other uncertainty reduction tactics and allowing them to be used in all the states in which they are applicable would cause a significant increase in the size of the model, possibly causing the optimal policy computation to be too slow for run-time decision. There may be many ways in which the calculations can be made tractable. For example, considering uncertainty reduction tactics only in the initial state (i.e., the state of the system at the time the decision is made) would reduce the size of the problem.

Instead of MDPs there may be other formalisms that could be used. In particular, partially observable Markov decision processes (POMDP) make no distinction between actions that change the system and actions that reduce uncertainty [21]. The main challenge with POMDPs will be making their solution fast enough to make decisions at run time.

Execute. Once the *plan* activity selects the uncertainty reduction tactics that must be executed, the *execute* activity takes care of

[4] By *failure* here we mean failure to execute. A fault such as producing incorrect output would not be detected even if the output was captured by *monitor*.

[5] If there had been more than one uncertainty reduction tactic applicable to the current state, the policy would have selected the optimal.

executing them. For regular adaptation tactics, this would be done using operators that change the target system. In the case of uncertainty reduction tactics, the operators could likewise change the system, for example, to turn a sensor on, or could launch some process, such as the malware scan in the motivating example. However, there are other kinds of uncertainty reduction tactics that do not change the target system, but instead change the *monitor* activity (e.g., instantiating a probe). One way to effect these changes would be indirectly through the *knowledge*, from which *monitor* would read its configuration every time *execute* changes it. Another option would be to have operators, similar to those defined for the target system, available to directly change the *monitor* activity.

Knowledge. The knowledge shared by all the activities typically consists of models of the system and its environment. For uncertainty reduction, it is necessary to extend these models to capture information about the uncertainty (e.g., the probability of server A being compromised in the example of the previous section). Cámara et al. present a catalog of patterns to represent uncertainty in self-adaptive systems, which could be useful to capture different kinds of uncertainty in the *knowledge* [7]. Among system models, specifications of tactic execution impact deserve particular attention and require description languages like the one introduced in [9] to be extended with specifications of how the level of uncertainty about the pieces of information in other models are modified by the tactic's execution. These specifications will be required to enable reasoning about the trade-offs of employing uncertainty reduction tactics.

4 UNCERTAINTY REDUCTION TACTICS

In addition to extending the activities and knowledge available to a self-adaptive system, enabling uncertainty reduction entails having available a catalog of tactics to improve the characterization of current and future system and environment states at run time by reducing uncertainties associated with different sources.

This section illustrates the uncertainty reduction mechanisms available in domains in which self-adaptation is currently being applied, even if these mechanisms are not being systematically exploited as we propose in this paper. There have been several works on identifying and categorizing types and sources of uncertainty in the self-adaptive systems literature, attending to different criteria and level of detail [14, 29, 35]. To illustrate some example forms of uncertainty reduction, we employ a subset of the categories described by Esfahani and Malek, who identify uncertainties associated with different sources in self-adaptive systems [14].

4.1 Uncertainty due to simplifying assumptions

The analyses used in the *analysis* activity of MAPE-K may have simplifying assumptions that make the analysis faster or even tractable. For example, the need to adapt may be determined by simply comparing a property of the system (e.g., the average response time) with a threshold set by the requirements of the system, ignoring details such as predictions of how the environment will change, or how recent changes in the system or its environment could affect the ability of the system to continue meeting requirements. The uncertainty introduced by these simplifications could be reduced using more elaborate analyses. In some cases, the use of a different analysis may require using different models in the *knowledge* and the necessary changes to *monitor* to gather the information required by the analysis. Changing the analysis would also require that *execute* can make changes to *analysis*. This can be accomplished directly or indirectly in the same ways described in Section 3 for changing *monitor*.

Tactic: use analysis with higher fidelity. For example, the uncertainty due to a simple threshold-based analysis can be reduced by using analyses with higher fidelity such as layered queuing networks [27] and queuing Petri nets for performance analysis [23], or runtime quantitative verification to predict requirement violations [4].

4.2 Uncertainty due to noise

This uncertainty arises from the changes in the process being monitored or from sensing error, which result in different values for each observation taken.

Tactic: change the sampling rate. It is possible to create a confidence interval for the value of the monitored parameter based on the mean and variance of the observations. One way to control the width of the interval, and thus the uncertainty, is to adapt the sampling rate [1, 15, 39, 42].

4.3 Uncertainty due to model drift

Over time, the models used as part of the *knowledge* component can become inconsistent with the actual state of the system or the environment due to the decoupling between the adaptation manager and the target system, or because there are changes happening that are not initiated by the adaptation manager. The uncertainty arising from model drift could be managed by adding probes, using on-going machine learning [6], or by other mechanisms that can refresh the information in the model that could be stale.

Tactic: probe service/device not recently used. When a service (or device) has not been recently used, there is uncertainty as to whether the service is still available. Even if the service is available, its response time may vary overtime, for example, due to the load the service is experiencing. Sending a request to a service before it is actually needed can be used as a tactic to reduce these kinds of uncertainty [19].

Tactic: turn probe/sensor on. In some cases, the uncertainty is due to not having monitored something such as the presence or state of a component, or the value of a parameter. In such cases, the uncertainty can be reduced by turning a probe or a sensor on so that the unknown state or value can be monitored [1, 12, 41].

Tactic: chaos monkey. There are cases in which the uncertainty is caused by changes in the underlying system that may affect some quality of the system. For example, Netflix uses a tool called *Chaos Monkey* to test the resiliency of their system—which is modified several times a day—by randomly terminating virtual machines [3]. In addition to testing whether the system can recover from this injected failure, this technique allows collecting other information that may change over time, such as how long it takes to recover (i.e., returning to an acceptable state according to some measure of effectiveness for the system).

4.4 Uncertainty in the context

As the context in which a system executes changes, the characteristic of different entities in the system's environment, and the availability of different resources may change, introducing uncertainty. This can be reduced using additional probes or more specific approaches such as discovery mechanisms [17] and tactics to discover external resources [26]. In addition to the tactic below, *turn probe/sensor on* described above is useful for this kind of uncertainty.

Tactic: present a CAPTCHA. When a system is experiencing unusually high load due to a denial-of-service (DoS) attack, there is uncertainty regarding which requests come from humans interacting with the system, and which are part of the attack and executed by bots. A CAPTCHA [40] can be used as a tactic to tell whether a connection is originated by a human or by a bot [38].[6]

4.5 Uncertainty due to human in the loop

When humans are used as an effector [8, 10], there may be uncertainty about their availability or willingness to perform a task.

Tactic: poll humans for their state. Elicit from the humans their availability or willingness to perform a task in order to reduce the uncertainty associated with relying on them. This could be done through self-report or brain-computer interfaces [20, 28].

4.6 Uncertainty due to decentralization

In this case, no single subsystem or unit of the decentralized system has complete knowledge of the state of all the units. Decentralized models can gather information about other systems to reduce the uncertainty about their state using direct queries or mechanisms such as gossiping and proximity broadcasting [43].

Tactic: query other systems. An uncertainty reduction tactic could request that information actively instead of waiting for it to be received, as is done with the *query* interaction protocol used in multi-agent systems [32].

5 RESEARCH CHALLENGES AND CONCLUSION

Self-adaptive systems face uncertainty stemming from a variety of sources. Although several self-adaptation approaches are able to take uncertainty into account when making adaptation decisions, to date there are no systematic techniques to reduce the uncertainty when that is possible and cost-effective.

In this paper, we have introduced the concept of *uncertainty reduction* to manage uncertainty in self-adaptive systems. The example of a self-protecting system presented in Section 2 shows how uncertainty reduction decisions can be integrated with self-adaptation decisions, achieving better effectiveness than the baseline approach that just considers the uncertainty without managing it. Sections 3 and 4 made this observation more concrete by describing how support for uncertainty reduction could be added to a self-adaptation loop, and by providing examples of uncertainty reduction tactics, respectively.

Although we have given a description of the changes that would be needed in a self-adaption loop to realize this idea, and have described several uncertainty reduction tactics, there are several interrelated research challenges that remain to be addressed.

In order for the system to decide when to use an uncertainty reduction tactic, it is necessary to assess the trade-off between its benefit and cost. Therefore, the impact of each tactic has to be specified, as is done with regular adaptation tactics. How to do this in a generic way is not obvious. For instance, in our example of Section 2, impact of the tactic was relatively easy to specify because it removed the uncertainty completely, and the probability of the two possible outcomes was derived from the IDS report. However, considering a tactic that just reduces the uncertainty without completely eliminating it brings up the challenge of quantifying how much of a reduction the tactic could achieve. Uncertainty quantification techniques [36] could help with that.

A catalog of uncertainty reduction tactics is needed so that practitioners can adopt the ones relevant to their systems. The challenge here is how to document the tactics in a way that is not system specific while still containing useful information. The techniques used for documenting design patterns [16] could be used as a model to achieve that goal.

Finally, decision procedures that can decide when to use uncertainty reduction tactics are needed. These procedures should evaluate the trade-offs between the benefit and the cost of using these tactics. Furthermore, it has to be sufficiently fast to make these decisions at run time (this might rule out certain formal reasoning mechanisms such as POMDPs). Ideally, uncertainty reduction decisions should be integrated with the regular self-adaptation decision in a way that treats both kinds of tactics uniformly. In that way, a decision approach that considers a look-ahead decision horizon would be able to assess adaptation paths consisting both kinds of tactics. In the example of Section 2, an MDP solved with a probabilistic model checker was used to achieve that. However, it is not immediately clear whether including different uncertainty reduction tactics, possibly with different kinds of impacts, could be supported with such an approach.

These three research challenges are not independent. Therefore, progress will have to be made tackling the three of them simultaneously. We intend to make progress in our future work to realize the ideas of uncertainty reduction put forward in this paper.

ACKNOWLEDGMENTS

Copyright 2018 ACM. All Rights Reserved. This material is based on research sponsored by AFRL and DARPA under agreement number FA8750-16-2-0042. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the AFRL, DARPA or the U.S. Government. This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. DM18-0045

[6] Unlike the other tactics presented, this one not only reduces uncertainty, but also changes the connections between the system and its context. Still, this tactic improves the outcome of subsequent adaptation decisions, for example, by preventing the system from responding to a DoS attack as if it were an increase in user traffic.

REFERENCES

[1] C. Alippi, G. Anastasi, M. Di Francesco, and M. Roveri. 2009. Energy management in wireless sensor networks with energy-hungry sensors. *IEEE Instrumentation and Measurement Magazine* 12, 2 (apr 2009), 16–23.

[2] Cornel Barna, Hamoun Ghanbari, Marin Litoiu, and Mark Shtern. 2015. Hogna: A Platform for Self-Adaptive Applications in Cloud Environments. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 83–87.

[3] Ali Basiri, Niosha Behnam, Ruud de Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, and Casey Rosenthal. 2016. Chaos Engineering. *IEEE Software* 33, 3 (may 2016), 35–41.

[4] Radu Calinescu, Carlo Ghezzi, Marta Kwiatkowska, and Raffaela Mirandola. 2012. Self-adaptive software needs quantitative verification at runtime. *Commun. ACM* 55, 9 (sep 2012), 69.

[5] Radu Calinescu, Lars Grunske, Marta Kwiatkowska, Raffaela Mirandola, and Giordano Tamburrelli. 2011. Dynamic QoS Management and Optimization in Service-Based Systems. *IEEE Transactions on Software Engineering* 37, 3 (may 2011), 387–409.

[6] Radu Calinescu, Yasmin Rafiq, Kenneth Johnson, and Mehmet Emin Bakir. 2014. Adaptive Model Learning for Continual Verification of Non-functional Properties. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE '14)*. ACM, New York, NY, USA, 87–98.

[7] Javier Cámara, David Garlan, Won Gu Kang, Wenxin Peng, and Bradley Schmerl. 2017. *Uncertainty in Self-Adaptive Systems*. Technical Report CMU-ISR-17-110. Institute for Software Research, Carnegie Mellon University.

[8] Javier Cámara, David Garlan, Gabriel A Moreno, and Bradley Schmerl. 2016. Evaluating Trade-Offs of Human Involvement in Self-Adaptive Systems. In *Managing Trade-Offs in Self-Adaptive Systems*, Ivan Mistrik, Nour Ali, John Grundy, Rick Kazman, and Bradley Schmerl (Eds.). Elsevier.

[9] Javier Cámara, Antonia Lopes, David Garlan, and Bradley Schmerl. 2014. Impact Models for Architecture-Based Self-Adaptive Systems. In *Proceedings of the 11th International Symposium on Formal Aspects of Component Software (FACS2014)*. Bertinoro, Italy.

[10] Javier Cámara, Gabriel A. Moreno, and David Garlan. 2015. Reasoning about Human Participation in Self-Adaptive Systems. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 146–156.

[11] Javier Cámara, Wenxin Peng, David Garlan, and Bradley Schmerl. 2017. Reasoning about Sensing Uncertainty in Decision-Making for Self-Adaptation. In *Proceedings of the 15th International Workshop on Foundations of Coordination Languages and Self-Adaptive Systems (FOCLASA 2017)*.

[12] Paulo Casanova, David Garlan, Bradley Schmerl, and Rui Abreu. 2014. Diagnosing unobserved components in self-adaptive systems. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems - SEAMS 2014*. ACM Press, New York, New York, USA, 75–84.

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

[14] Naeem Esfahani and Sam Malek. 2013. Uncertainty in self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems II*, Rogério de Lemos, Holger Giese, Hausi A. Müller, and Mary Shaw (Eds.). Lecture Notes in Computer Science, Vol. 7475. Springer Berlin Heidelberg, 214–238.

[15] Mohamed Medhat Gaber and Philip S. Yu. 2006. A framework for resource-aware knowledge discovery in data streams. In *Proceedings of the 2006 ACM symposium on Applied computing - SAC '06*. ACM Press, New York, New York, USA, 649.

[16] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1993. Design Patterns: Abstraction and Reuse of Object-Oriented Design. *Lecture Notes in Computer Science* 707 (1993), 406–431.

[17] David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley Schmerl, and Peter Steenkiste. 2004. Rainbow: architecture-based self-adaptation with reusable infrastructure. *Computer* 37, 10 (oct 2004), 46–54.

[18] Simos Gerasimou, Radu Calinescu, and Alec Banks. 2014. Efficient runtime quantitative verification using caching, lookahead, and nearly-optimal reconfiguration. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems - SEAMS 2014*. ACM, New York, New York, USA, 115–124.

[19] Julia Hielscher, Raman Kazhamiakin, Andreas Metzger, and Marco Pistore. 2008. A framework for proactive self-adaptation of service-based applications based on online testing. In *1st European Conference on Towards a Service-Based Internet*, Vol. 5377. Springer Berlin Heidelberg, 122–133.

[20] Shihong Huang and Emmanuelle Tognoli. 2014. Brainware: synergizing software systems and neural inputs. In *Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014*. ACM Press, New York, New York, USA, 444–447.

[21] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101, 1 (1998), 99–134.

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

[23] S. Kounev and A. Buchmann. [n. d.]. Performance modelling of distributed e-business applications using Queuing Petri Nets. In *2003 IEEE International Symposium on Performance Analysis of Systems and Software. ISPASS 2003.* IEEE, 143–155.

[24] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. 2014. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing* 17, Part B (oct 2014), 184–206.

[25] Marta Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: verification of probabilistic real-time systems. In *23rd international conference on Computer Aided Verification*. Springer-Verlag, 585–591.

[26] Grace A. Lewis and Patricia Lago. 2015. A Catalog of Architectural Tactics for Cyber-Foraging. In *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures - QoSA '15*. ACM Press, New York, New York, USA, 53–62.

[27] Marin Litoiu and Cornel Barna. 2013. A performance evaluation framework for Web applications. *Journal of Software: Evolution and Process* 25, 8 (aug 2013), 871–890.

[28] Eric Lloyd, Shihong Huang, and Emmanuelle Tognoli. 2017. Improving Human-in-the-Loop Adaptive Systems Using Brain-Computer Interaction. In *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 163–174.

[29] S Mahdavi-Hezavehi, P Avgeriou, and D Weyns. 2016. A Classification of Current Architecture-based Approaches Tackling Uncertainty in Self-Adaptive Systems with Multiple Requirements. In *Managing Trade-offs in Adaptable Software Architectures*, Ivan Mistrik, Nour Ali, Rick Kazman, John Grundy, and Bradley Schmerl (Eds.). Elsevier, Chapter 3, 45 – 77.

[30] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. 2015. Proactive self-adaptation under uncertainty: a probabilistic model checking approach. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*. ACM Press, New York, New York, USA, 1–12.

[31] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. 2016. Efficient Decision-Making under Uncertainty for Proactive Self-Adaptation. In *2016 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, Wuerzburg, Germany, 147–156.

[32] Stefan Poslad. 2007. Specifying protocols for multi-agent systems interaction. *ACM Transactions on Autonomous and Adaptive Systems* 2, 4 (nov 2007).

[33] M. L. Puterman. 2002. Dynamic programming. *Encyclopedia of Physical Science and Technology* 4 (2002), 673–696.

[34] M. L. Puterman. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, Ltd.

[35] Andres J Ramirez, Adam C Jensen, and Betty H C Cheng. 2012. A taxonomy of uncertainty for dynamically adaptive systems. In *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 99–108.

[36] Christopher J Roy and William L Oberkampf. 2011. A comprehensive framework for verification, validation, and uncertainty quantification in scientific computing. *Computer Methods in Applied Mechanics and Engineering* 200, 25 (2011), 2131–2144.

[37] Mazeiar Salehie and Ladan Tahvildari. 2009. Self-adaptive software: Landscape and Research Challenges. *ACM Transactions on Autonomous and Adaptive Systems* 4, 2 (may 2009), 1–42.

[38] Bradley Schmerl, Javier Cámara, Jeffrey Gennari, David Garlan, Paulo Casanova, Gabriel A. Moreno, Thomas J. Glazier, and Jeffrey M. Barnes. 2014. Architecture-based self-protection. In *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security - HotSoS '14*. ACM Press, New York, New York, USA, 1–12.

[39] Niels L. M. van Adrichem, Christian Doerr, and Fernando A. Kuipers. 2014. OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks. In *2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, vanAdrichem2014, 1–8.

[40] Luis von Ahn, Manuel Blum, and John Langford. 2004. Telling humans and computers apart automatically. *Commun. ACM* 47, 2 (feb 2004), 56–60.

[41] David L Wells and Paul Pazandak. 2001. Taming cyber incognito: Tools for surveying Dynamic/Reconfigurable software landscapes. In *Working Conference on Complex and Dynamic Systems Architectures, Brisbane, Australia*. 13–24.

[42] Wenhong Ma, Changcheng Huang, and J. Yan. 2004. Adaptive sampling for network performance measurement under voice traffic. In *2004 IEEE International Conference on Communications (IEEE Cat. No.04CH37577)*. IEEE, 1129–1134 Vol.2.

[43] Danny Weyns, Sam Malek, and Jesper Andersson. 2010. On decentralized self-adaptation. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems - SEAMS '10*. ACM Press, New York, New York, USA, 84–93.

[44] Eric Yuan, Naeem Esfahani, and Sam Malek. 2014. A Systematic Survey of Self-Protecting Software Systems. *ACM Transactions on Autonomous and Adaptive Systems* 8, 4 (jan 2014), 1–41.