Software and Systems Modeling (2022) 21:1277–1294
https://doi.org/10.1007/s10270-022-01037-6

EXPERT VOICE

The uncertainty interaction problem in self-adaptive systems

Javier Cámara[1] · Javier Troya[1] · Antonio Vallecillo[1] · Nelly Bencomo[2] · Radu Calinescu[3] · Betty H. C. Cheng[4] · David Garlan[5] · Bradley Schmerl[5]

Received: 11 April 2022 / Accepted: 19 August 2022 / Published online: 17 August 2022

Abstract
The problem of mitigating uncertainty in self-adaptation has driven much of the research proposed in the area of software engineering for self-adaptive systems in the last decade. Although many solutions have already been proposed, most of them tend to tackle specific types, sources, and dimensions of uncertainty (e.g., in goals, resources, adaptation functions) in isolation. A special concern are the aspects associated with uncertainty modeling in an integrated fashion. Different uncertainties are rarely independent and often compound, affecting the satisfaction of goals and other system properties in subtle and often unpredictable ways. Hence, there is still limited understanding about the specific ways in which uncertainties from various sources interact and ultimately affect the properties of self-adaptive, software-intensive systems. In this SoSym expert voice, we introduce the *Uncertainty Interaction Problem* as a way to better qualify the scope of the challenges with respect to representing different types of uncertainty while capturing their interaction in models employed to reason about self-adaptation. We contribute a characterization of the problem and discuss its relevance in the context of case studies taken from two representative application domains. We posit that the Uncertainty Interaction Problem should drive future research in software engineering for autonomous and self-adaptive systems, and therefore, contribute to evolving uncertainty modeling towards holistic approaches that would enable the construction of more resilient self-adaptive systems.

Keywords Uncertainty · Modeling · Self-adaptation · Assurances

1 Introduction

Complex systems in which software contributes essential influences to the design, construction, deployment, and evolution of the system, commonly referred to as *software-intensive systems*, are increasingly relied upon to support tasks that are typically characterized by a high degree of *uncertainty* in different contexts.

Frequently, uncertainty is unavoidable. Having a means to explicitly represent uncertainty can significantly help avoid making decisions associated with a high level of unpredictability, which can lead to actions involving major losses

Javier Troya, Antonio Vallecillo, Nelly Bencomo, Radu Calinescu, Betty H.C. Cheng, David Garlan and Bradley Schmerl authors contributed equally to this study.

✉ Javier Cámara
jcamara@uma.es

Javier Troya
jtroya@uma.es

Antonio Vallecillo
av@uma.es

Nelly Bencomo
nelly.bencomo@durham.ac.uk

Radu Calinescu
radu.calinescu@york.ac.uk

Betty H. C. Cheng
chengb@msu.edu

David Garlan
garlan@cs.cmu.edu

Bradley Schmerl
schmerl@cs.cmu.edu

[1] ITIS Software, University of Málaga, Málaga, Spain

[2] Department of Computer Science, Durham University, Durham, UK

[3] Department of Computer Science, University of York, York, UK

[4] Department of Computer Science and Engineering, Michigan State University, East Lansing, USA

[5] Institute for Software Research, Carnegie Mellon University, Pittsburgh, USA

or costs, i.e., the so-called *cost of being wrong* [38]. Examples of such situations are a facial recognition system that misidentifies injured people in a search and rescue operation due to a natural disaster, an automatic triage application that misclassifies an emergency room patient as low risk when she is actually critically ill, or the automatic stock management system that decides to make a substantial investment based solely on the confidence of a computer system prediction, but without considering its associated uncertainty, can all lead to disastrous consequences if the wrong decision is made. Having a measure of the degree of uncertainty associated with a resulting decision can be useful, for example to rule out any decision whose degree of uncertainty is above a certain threshold [5,24].

Addressing these situations by mitigating the effects of uncertainty in software-intensive systems has been one of the focal points of self-adaptation [9,22], which is regarded as a promising way to engineer in an effective manner systems that are *resilient* to run-time changes despite the different uncertainties in their execution environment (induced by, e.g., resource availability, interaction with human actors), goals, or even in the system itself (e.g., faults, noisy sensors, machine learning-based components).

During the last decade, the research community has made an important effort in supporting the analysis and management of self-adaptive systems under uncertainty. In particular, several taxonomies of uncertainty have been proposed [16,29,35,36], and a substantial body of work exists on methods to manage uncertainty [21]. These methods are able to *individually* detect, represent, and mitigate uncertainties from various sources, which may have distinctive effects on the satisfaction of both functional and non-functional requirements. However, these *uncertainties are rarely independent and often compound*, affecting the satisfaction of goals and other system properties in subtle and often unpredictable ways, as we illustrate in the remainder of this article.

We believe that there is still limited understanding about the specific ways in which uncertainties of different types and from various sources interact and, ultimately, affect the properties of self-adaptive, software-intensive systems. Representing the interactions of such combined uncertainties, analyzing the impact of their emergent effects on the satisfaction of requirements, and mitigating them remain open challenges that are the focus of this article.

The remainder of this paper is structured as follows. Section 2 gives an introduction to self-adaptive systems, explains the uncertainty sources identified in this domain, and presents a classification of uncertainties. Then Sect. 3 exemplifies the uncertainty interaction problem in systems from two application domains, where several types of uncertainty interactions are illustrated. Section 4 outlines research challenges that should be addressed in order to deal with the uncertainty interaction problem. Finally, Sect. 5 concludes the paper.

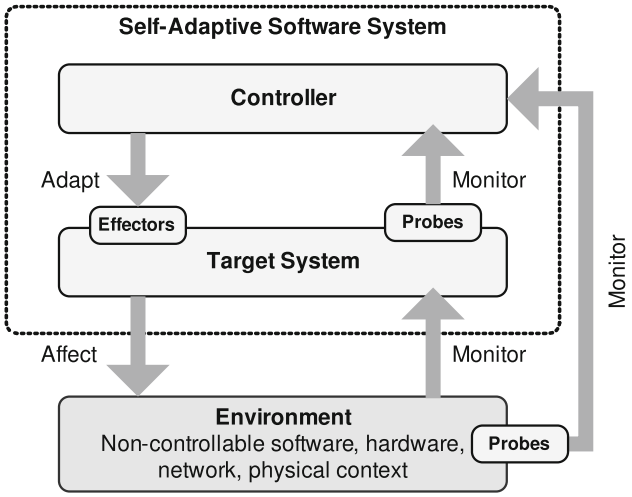

Fig. 1 Self-adaptive system

2 Background

In this section, we first present some general background about self-adaptive systems. Next, we introduce a categorization of the sources of uncertainty in self-adaptive systems informed by the work of Mahdavi-Hezavehi et al. [29], followed by a taxonomy of uncertainty classified according to its representation in software models [44].

2.1 Self-adaptive systems

What distinguishes a self-adaptive system from any other system is its ability to continuously deliver its service despite changes that may occur in the system, its environment, or its goals. A key component that enables self-adaptive systems to handle changes at run-time is a *controller* or *self-adaptive layer* (Fig. 1) that implements a set of *adaptation functions* and that relies on a feedback loop for managing adaptations [9]. Controllers execute actions via effectors or actuators on the *target system*, based on information monitored by probes or sensors both from the target system and its *environment*, which consists of all non-controllable elements that determine the operating conditions of the system (*e.g.*, hardware, network, physical context, etc.).

In the context of complex software-intensive systems, these controllers are usually built based on patterns such as MAPE-K [25], which implements the traditional *sense-plan-act* architectures and includes four distinct operational stages:

- *Monitor*: monitors the target system and environment through *sensors* that provide information about the value of relevant variables.
- *Analyze*: decides whether the current state demands adaptation.

- *Plan*: if adaptation is required, decides the best way to adapt the system.
- *Execute*: applies a sequence of control or adaptation actions through system-level *effectors* (or actuators).

Key concerns with respect to the run-time behavior of self-adaptive systems are related to their non-functional attributes that include performance, cost, availability, reliability, and safety. Self-adaptive systems typically operate in environments that feature high-levels of uncertainty that emerge from different sources and have a remarkable influence on the satisfaction of system goals and other system properties. In the following section, we identify a number of those sources.

2.2 Uncertainty sources in self-adaptive systems

Several papers have identified types of uncertainty in self-adaptive systems [8,16,45]. Among them, the work by Mahdavi-Hezavehi et al. [29] defines a comprehensive and structured organization that identifies and categorizes sources of uncertainty. The study identifies five dimensions of uncertainty: location, source, nature, level/spectrum, and emerging time. For our purposes, we focus on source and location (i.e., the class of uncertainty source), abstracting away from other dimensions that are not essential to illustrate the ideas in this paper. Next, we provide details of these two dimensions of uncertainty.

Location. The place in which uncertainty emerges in the self-adaptive system. Identified locations[1] are:

1. *Model*: Models that the self-adaptive system employs (typically for decision-making). One example might be the abstraction of some aspect of the real system that is not represented in its model, which induces epistemic uncertainty.
2. *Adaptation Functions*: Functionalities that a self-adaptive loop performs. An example is the uncertainty caused by faulty sensors of the adaptive system.
3. *Goals*: Goals that the self-adaptive loop uses to manage the system. An example is not fully anticipating changing goals in the future.
4. *Environment*: Context (including interactions with human actors) in which the system is running (e.g., the uncertainties induced by the behavior of a human-in-the-loop, which is not deterministic).
5. *Resources*: Components needed by the self-adaptive system to operate. An example is uncertainty from changes in resource availability.
6. *Managed System*: Subsystem being managed by the managing subsystem in the self-adaptive system. An example is uncertainty caused by the complexity of the managed subsystem.

Source. It represents specific sources of uncertainty within the locations described above. Table 1 describes the different sources of uncertainty identified in [29].

In addition to the information presented in the table, Fig. 2 shows a UML class diagram that represents the six uncertainty sources (classes shaded in light brown color in the diagram) and their relationships, which in this case represent the possible interactions between them. For example, the uncertainty of self-adaptive behavior may be aggravated by a goal uncertainty when combined. Suppose that the threshold used to decide about a change in behavior is not cleanly defined because it depends on sensor values, which are imprecise. Suppose as well that the utility functions, which define the objectives, are neither precisely defined because stakeholders do not have a precise idea about in what contexts they should prioritize performance over safety (the information can be learned during run-time [3,40]). Similarly, the decision to change a resource may be much more difficult when it is not clear if the resource is available or not, and such a decision is based on imprecise functions. In the following sections we will describe specific examples of these interactions in two scenarios that we have chosen to illustrate some of the problems that can emerge when combining uncertainties in self-adaptive systems.

2.3 Representing uncertainty

Quoting DeMarco [13],"You can't control what you can't measure." In order to measure uncertainty, first we need to represent it. The purpose of explicitly representing uncertainty is twofold: a software engineer who models or simulates a system needs to capture the relevant characteristics of uncertainty in a suitable way so they can be made explicit and later analyzed; while a systems engineer analyses uncertainty to quantify it, reduce it, or mitigate its effects [2,31].

Scientists and engineers already know how to deal with uncertainty in many of its forms (objective, subjective, epistemic, aleatory) [43], using different approaches such as mathematical and numerical models [32], probabilities [3, 18], Fuzzy set theory [39,48], variability analysis [42] or risk assessment [37], among others.

The representation of uncertainty depends largely on its nature. Expressing the precision of a physical measurement is neither the same as expressing the degree of belief that a person has about a given fact, nor the design decisions that a software engineer must contemplate under imprecise user requirements. Different types of uncertainty require different notations, underlying logics, and inference mechanisms

[1] In the remainder of this article, we use the term *uncertainty source* informally to refer both to uncertainty source groups (locations) and specific sources of uncertainty.

Table 1 Sources of uncertainty (from [29])

Source group (location)	Uncertainty source	Description
Model	Abstraction	Uncertainty caused by omitting certain details and information from models for the sake of simplicity
	Incompleteness	Uncertainty caused by parts (of models, mechanisms, etc.) that are knowingly missing because of a lack of (current) knowledge
	Model drift	Uncertainty caused by a discrepancy between the state of models and the represented phenomena
	Different sources of information	Uncertainty caused by differences between the representations of information provided by different sources of information. Uncertainty may be due to different representations of the same information, or result of having different sources of information, or both
	Complex models	Uncertainty caused by complexity of runtime models representing managed sub systems
Adaptation functions	Variability space of adaptation	Uncertainty caused by the size of the variability space that the adaption functions need to handle. This type of uncertainty arises from striving to capture the whole complex relationship of the system with its changing environment in a few architectural configurations which is inherently difficult and generates the risk of overlooking important environmental states [5]
	Sensing	Uncertainty caused by sensors that are inherently imperfect
	Effecting	Uncertainty caused by actuators that are inherently imperfect
	Automatic Learning	Uncertainty caused by machine learning techniques of which the effects may not be completely predictable
	Decentralization	Uncertainty due to decision making by different entities of which the effects may not be completely predictable
	Changes in adaptation mechanisms	Uncertainty due to required dynamicity of adaptation infrastructure to maintain its relevance with respect to the changing adaptation goals (Villegas, Tamura, Müller, Duchien, and Casallas, 2013)
	Fault localization and identification	Uncertainty caused by inaccurate localization and identification of faults in the managed system
Goals	Goal dependencies	Dependencies between goals, in particular quality goals, may not be captured in a deterministic manner, which causes uncertainty
	Future goal changes	Uncertainty due to potential changes of goals that could not be completely anticipated
	Future new goals	Uncertainty due to the potential introduction of new goals that could not be completely anticipated
	Goal specification	Uncertainty due to lack of deterministic specifications of quality goals
	Outdated goals	Uncertainty caused by overlooking outdated goals
Environment	Execution context	Uncertainty caused by the inherent unpredictability of execution contexts
	Human in the loop	Uncertainty caused by the inherent unpredictability of human behavior
	Multiple ownership	Uncertainty caused by lack of proper information sharing, conflicting goals, and decision making policies by multiple entities that own parts of the system
Resources	New resources	Uncertainty caused by availability of new resources in the system
	Changing resources	Uncertainty caused by dynamicity of resources in the system
Managed system	System complexity and changes	Uncertainty caused by complexity and dynamicity of nature of the managed system

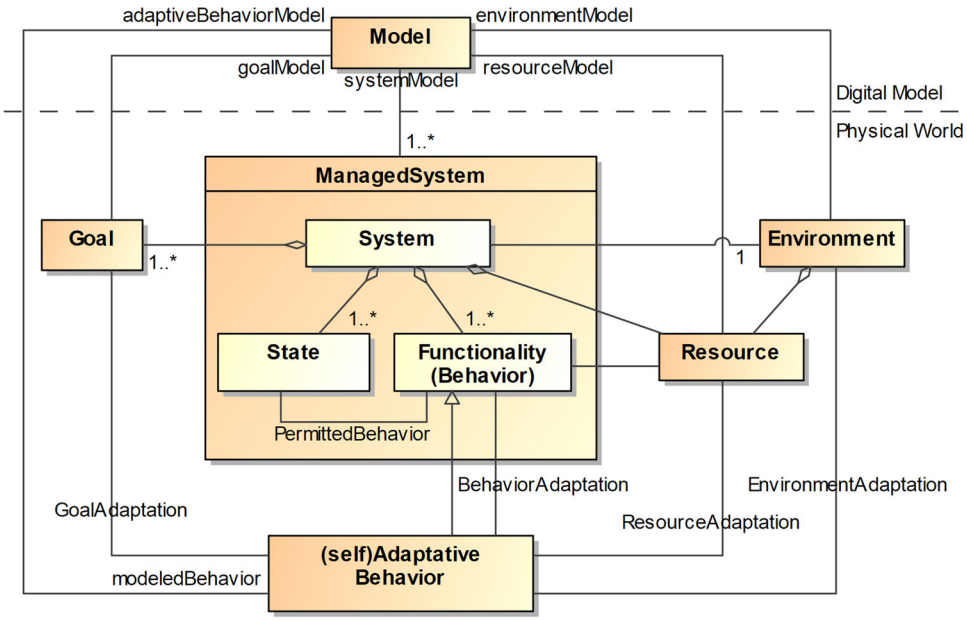

Fig. 2 Uncertainty sources and their relationships

to reason about them, for example, to analyze its properties or how an uncertainty propagates. A classification of uncertainty in terms of the notations used to represent it in software models was presented in [44], which is summarized below. This classification is relevant in our context since we need to represent uncertainties in order to be able to study their interaction and quantify them. Their combination may be easier if they are similar in nature, admit the same representations and are amenable to similar reasoning mechanisms. Similarly, studying the interaction of two uncertainties of very different nature will pose more complex challenges. This classification defines six dimensions:

Measurement uncertainty. This is an aleatory uncertainty that refers to a set of possible states or outcomes of a measurement [23]. For example, the distance to an object can be expressed as 123 ± 0.1 cm, assuming our sensor has a precision of 1 mm. Similarly, a Boolean variable can be best represented by a number in the range [0,1] representing the probability that the variable is true.

Occurrence uncertainty. This uncertainty refers to the degree of belief that we have on the actual existence of an entity, i.e., the real entity that a model element represents. It is normally represented in terms of probabilities [12], fuzzy set theory [39] or subjective logic [24]. For instance, we can be only 70% confident that the required resource will be available when actually needed.

Design uncertainty. This type of epistemic uncertainty refers to a set of possible decisions or system design options. It captures the usual uncertainty that the developer has about the system design, which may be different depending on the conditions the system may face during its operation and the expected requirements by its intended users. This information is normally unknown during the early analysis phase, but heavily influences the system design. Variability models are commonly used to represent this type of uncertainty [17].

Behavior uncertainty. This epistemic uncertainty refers to the lack of knowledge about the behavior of the system or its environment. It is common in self-adaptive [11,46] or uncertainty-aware systems [19,47], whose operating environments are unknown or may exhibit uncertain behaviors; for example, a robot operating on Mars, or an application whose users may have erratic or random behaviors [29,33].

Belief uncertainty. This epistemic uncertainty occurs when belief agents are uncertain about how the system has been modeled or about the system itself. For example, two different users may have different degrees of trust on the measurements of the sensors depending on their previous experiences with them, and therefore will interpret their values differently. Again, probabilities, fuzzy set theory, or subjective logic are used to represent this type of subjective uncertainty (e.g., in self-adaptive system requirements [1]).

Spatio-temporal uncertainty. This type of epistemic uncertainty refers to the lack of certainty about the geographical or physical location of a system, its elements or its environment, or about properties that relate to the timing of events in the system or its environment. While measurement uncertainty expresses possible variations of a measured value, and is of an aleatory and objective nature, spatiotemporal uncertainty implies vagueness and incompleteness, and is of epistemic and subjective nature—e.g., stating that an archaeological site is located "somewhere" in Northern Europe, or that an event happened "a bit later" than another.

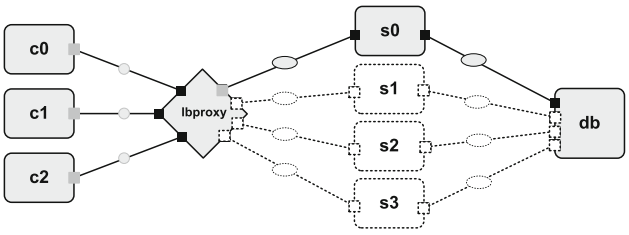

Fig. 3 Znn.com architecture

3 The uncertainty interaction problem in self-adaptive systems

In this section we present two different scenarios of self-adaptive systems and describe how the six types of sources of uncertainty explained in Sect. 2.2 can take place in each of them. We also explain how the uncertainty interaction problem may occur in these scenarios. Finally, we exemplify uncertainty interactions, such as those that take place between model and adaptation functions, model and environment, as well as environment and adaptation functions, among others.

3.1 ZNN news

Znn.com is a simple news service that uses MAPE-K (embodied by the Rainbow framework to deal with varying workloads—including *slashdot effect* and DoS attacks) through different tactics [41]. As Fig. 3 indicates, Znn.com has several servers (some of which are inactive), a load balancer, and a database. Rainbow monitors the load balancer, servers, and the database to update the different models of the system, environment, etc. to make decisions. In order to adapt to changing workloads, the self-adaptive layer executes different tactics such as activating servers, deny-listing clients, enabling a CAPTCHA, or changing the quality of the contents served, among others.

3.1.1 Sources of uncertainty

Let us consider a situation in which Znn.com, receives a spike in workload with a high request arrival rate and has to decide which of the tactics to trigger (if any), in order to continue to satisfy system goals. We may face different uncertainties in such a situation. Some examples are provided in Table 1, classified according to their source.

- *Model*: The different models in the knowledge base of the self-adaptive layer of Znn.com are also potential sources of uncertainty. For example, the abstraction level of system and environment properties (e.g., coarse-grained discretization of numerical variables like the request arrival rate at the load balancer in the environment model), different representations of the same information (e.g., there can be discrepancies between the response time as directly monitored by the system and the one calculated based on CPU load and queue length), or the modeling paradigm used (e.g., queuing models and continuous-time Markov chains introduce error when modeling real-world phenomena [34]).
- *Adaptation functions*: The exact outcome of executing a given adaptation tactic (e.g., activating a new server) is unknown, in terms of precise improvements on throughput or response time. Sensing is also imperfect so measurements taken, e.g., at the load balancer to gauge the request arrival rate, may be imprecise (averaging windows are typically employed to mitigate quick fluctuations), or even outdated by the time they are incorporated in models. The time that it takes to execute an adaptation tactic (i.e., its *latency*) is also subject to uncertainty, e.g., the time that it takes to activate a server can suffer remarkable fluctuations depending on environment conditions.
- *Goals*: In Znn.com, dependencies among goals are not captured explicitly. Instead, the selection of adaptations to satisfy the set of extra-functional goals (i.e., cost minimization, user experience optimization, security) is driven by utility functions that do not clarify for instance under what conditions security has priority over cost, and vice versa. In some cases, a spike in incoming traffic due to a DoS attack can be handled with security tactics, such as a deny list for potentially malicious users, or just adding more resources at the expense of increased cost when the priority is to maintain adequate service provision for legitimate users, rather than eliminating potential attackers from the system. Currently, goals are fixed in MAPE-K but this constraint is likely to be suboptimal.
- *Environment*: the evolution of the request arrival rate (e.g., whether it is going up, down, or remains stable) can be predicted in some cases, but only to some degree of certainty, e.g., using a time series predictor [30]. This is important for anticipating to usage peaks when more resources may be needed. The nature of the access of clients to the system (i.e., whether they are legitimate clients or bots attempting to perform a DoS attack) is unknown. We need this information to decide whether preventive measures, such as the use of captchas, are worth the potential inconvenience to most users.
- *Resources*: servers may fail, and considering their expected failure rate may provide more realistic estimates when sizing the system. However, predictions based on high uncertainty can increase the number of servers required, thereby unnecessarily increasing the overall costs. Similarly, the availability of additional servers to activate to spread workload across more servers may only be known with some degree of certainty, because they

may not be available at all times. Their response times and performance can also vary, introducing new sources of uncertainty when predicting the overall system performance.

- *Managed system*: This uncertainty is caused by the complexity and dynamicity of the managed system, which hinders the estimations of its behavior. The system and its parts may also evolve, incorporating new elements whose behavior was not considered when the system was designed. These parts may also fail or behave in erratic or unexpected manners, e.g., the performance of the database can degrade if the number of records goes above the limit for which it was originally intended to store, or can suffer attacks or intermittent failures. These uncertainties may amplify the overall system uncertainty when combined with others.

In this paper, we are interested in the cases where two or more of these uncertainties are combined, and the effects of the interactions between them. The following subsections describe particular scenarios that serve to illustrate these situations and their effects.

3.1.2 Uncertainties due to model and adaptation functions

In the MAPE-K loop implemented for Znn.com, the analysis stage is in charge of detecting if the triggering of an adaptation mechanism is needed due to invariant violation. One typical example is the invariant stating that the current system response time r should always be below some threshold $R_{\max}$ specified in one of the system models. In this setting, we could observe the following situations related to the interference of sources of uncertainty with respect to the correct operation of the analysis:

- The imperfect sensing of the response time property could yield values within some range $[\hat{r}_{min}, \hat{r}_{\max}]$ that contains the ground truth value r. If the observed value is above threshold ($\hat{r} \geq R_{\max}$), but the real value is not ($r < R_{\max}$), this will lead to a false positive that will trigger an adaptation planning and execution cycle when it is not really needed, increasing the cost of operating the system without need. The symmetric case can be given when $\hat{r} < R_{\max} \leq r$, meaning that the adaptation cycle will not be triggered even if it is really needed, causing an unnecessary degradation of performance that could have been addressed otherwise.
- The coarse-grained discretization of the response time property in the managed system model can also result in undesired adaptation triggers or the lack of required adaptations when the discretized value r_d of the property is on the other side of the threshold, when compared to the ground truth value r. Hence, if response time is discretized with a granularity of $\eta_r = 200$ ms, response times measured where $\|r - R_{\max}\| \leq 200$ could be problematic, depending on the concrete discretization scheme used.

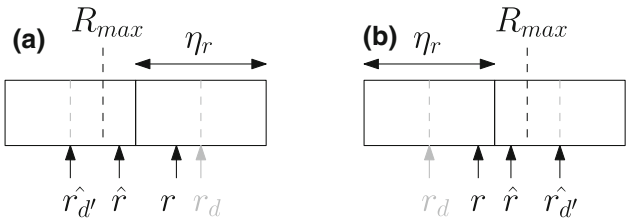

Fig. 4 Model-adaptation function interaction: **a** causing execution of spurious adaptation **b** preventing execution of required adaptation

Avoiding the problems derived from the two situations described above could involve the explicit modeling of the uncertainty induced by the imperfection in the sensing and the discretization of the response time property values.

However, these two sources of uncertainty can interact with each other in multiple ways. Consider for instance the situation illustrated in Fig. 4a, where the ground truth value of the variable r is observed with some error. Both r and the observed value $\hat{r}$ lie below threshold $R_{\max}$. In the figure, dashed gray lines represent the values that the variable can take in the discrete abstraction of the system model. Any real value observed in the surrounding box will be snapped to that discrete value. We observe that even if the observed value of the variable $\hat{r}$ contains some error, this error on its own would not be enough to trigger an undesired adaptation. However, due to the discretization of the observed variable values, $\hat{r}$ will be snapped to $\hat{r}_{d'}$, which is above $R_{\max}$, triggering a spurious adaptation. Note that without the error induced by observation, the discretization process on its own would not have been enough to trigger this adaptation, given that r would have been snapped to r_d, which is below $R_{\max}$. Conversely, we have the situation illustrated in Fig. 4b, in which both the ground truth and the observed value of the variable lie above the threshold. However, the discretization process snaps the value of $\hat{r}$ to $\hat{r}_{d'}$, preventing the triggering of adaptation in a situation in which it would have been required. Analogously to the situation described in (a), neither the discretization process nor the observation error on their own would have been enough to prevent the execution of the required adaptation. Instead, it is the compound effect of both sources of uncertainty that causes the undesired situation.

Addressing this issue. This case could be treated by representing numbers with their associated uncertainty [23,44]. In this context, comparison between numbers is no longer a Boolean operation but returns a probability that expresses the likelihood (or confidence) of being one number less than, equal to, or greater than the other [4]. Given that the sources

of uncertainty of the two facts in this example are independent (accuracy of r and its snapping $\hat{r}_{d'}$), their combination can be computed by multiplying the confidence of the comparison between the uncertain response time and the threshold, with the degree of uncertainty caused by the discretization process. Then the decision can be made considering not only whether $r > R_{\max}$ but also its associated degree of uncertainty, discarding those decisions where the uncertainty is above a certain threshold.

3.1.3 Uncertainties due to goal and adaptation functions

Similarly to other self-adaptive systems, the selection of an adaptation in Znn.com is driven by a utility function U that balances the tradeoffs among multiple concerns. These functions are often encoded as a linear combination of terms, where each term is a utility function that captures a given concern (e.g., performance, cost), moderated by a weight that specifies user priorities:

$$U = w_p \cdot u_p(r) + w_c \cdot u_c(c) \tag{1}$$

In Equation 1, $u_p, u_c : \mathbb{R}_{\geq 0} \rightarrow [0, 1]$ are utility functions for the concerns of performance and cost, which map a response time r and a cost c, respectively, to a utility value. Weights w_p and w_c, which sum up to 1 capture the relative importance of each term. Selection of a specific adaptation, designated by the term *adaptation strategy*, is carried out by analyzing the anticipated effect of the different strategies available on the value of U, and choosing the one that maximizes that value. This analysis relies on models in the knowledge base of the MAPE-K loop that capture the expected impact of available adaptation actions (or *adaptation tactics*), which are the building blocks of adaptation strategies, on the qualities of the system. A simple model of expected adaptation tactic impact could include entries similar to the following one: ActivateServer $[r : -500, c : +5]$. This entry captures that the tactic ActivateServer is expected to reduce response time r by 500 ms and to increase the operating system cost in 5 USD/hr. Of course, this class of model is simple and does not explicitly capture the obvious uncertainties that concern, for instance, the variability of the execution context (e.g., response time reduction does not behave in a linear manner, and the activation of more servers does not always result in the same reduction of 500 ms.).

Moreover, additional sources of uncertainty may interfere with the adaptation strategy. First, it may be the case that there are no servers available when they are needed (for instance, because the monitors employed for service availability are not completely reliable), so even when a given strategy provides the best estimate, we cannot be sure about its real outcome, or even if it can be applied. Second, in practice the utility functions u_p and u_c of Equation (1) have associated uncertainties, the same as the corresponding weights w_p and w_c. This means that U should be more faithfully represented accompanied by an associated uncertainty, i.e., given in terms of $U \pm d$, where d represents its standard deviation [23]. For example, $U_1 = 0.7 \pm 0.3$, which is more informative than a crisp value of $U_1 = 0.7$. Thus, the comparison between strategies should be made by comparing uncertain instead of crisp values, which will give an indication of the confidence associated with the resulting utility value. This may provide very useful information. For example, if $U(s_1) = 0.70$ and $U(s_2) = 0.65$ are two utility function values that correspond to two competing alternative strategies s_1 and $s2$, the adaptive system will select the former because clearly $U(s_1) > U(s_2)$. However, if we know that $U(s_1) = 0.7 \pm 0.3$ and $U(s_2) = 0.65 \pm 0.2$, then the situation is not as clear since the probability that $U(s_1) > U(s_2)$ is only 0.15. In fact, in this case it would be better to select s_2 because it has less associated uncertainty.

There is also another problem due to uncertainties of epistemic nature, i.e., those caused by lack of knowledge, and where probabilistic logic is not expressive enough to capture them and, therefore, other belief logics are often used instead. For example, suppose two stakeholders who assign different levels of confidence to the values of the utility functions because their trust in the sensors' values is different. Imagine that John trusts the sensors that produce the values of u_p and u_c, but Eva is aware that they have been running for too long, and thus their performance measures are imprecise. Therefore, she assigns different degrees of belief to their measurements and hence to the value of u_p, whose uncertainty is altered according to the confidence of Eva, but maintained in the case of John. How to combine two types of uncertainty, namely measurement uncertainty and belief uncertainty, is not easy. In this case, Subjective logic [24] could be used to represent opinions, combine them with the measurement values, and thereby help the two engineers reach a consensus using a fusion operator [5].

3.1.4 Uncertainties due to adaptation functions

Different sources of uncertainty within the adaptation functions of a self-adaptive system can also interact among themselves. Consider for instance the uncertainty in the *latency* of the adaptation tactics of a self-adaptive system (i.e., the time that spans between the triggering of the execution of the adaptation action and the time instant in which its effects take place), and the imperfect sensing of system variables.

In the context of Znn.com, the adaptation tactic ActivateServer mentioned in Sect. 3.1.3 has a latency associated with the time that it takes to boot up a new server and for it to start processing incoming requests. Of course, there is an uncertainty associated with that latency because under different

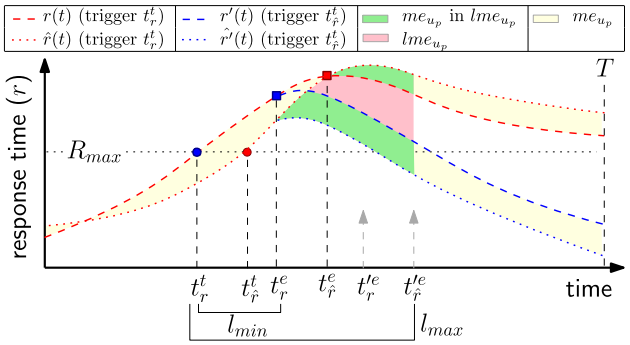

Fig. 5 Adaptation function uncertainty interaction: imperfect sensing and uncertainty in latency

execution and network conditions, spinning up a new server may take different amounts of time that range between a few seconds and several minutes [20]. Ignoring such uncertainty in the latency can result in inefficiencies due to the system performing a suboptimal sequence of adaptations. For example, the system may adapt to handle a transient change in workload, only to have to adapt back to the previous configuration moments later. If the cost of performing those two adaptations is higher than their benefit, then it would be better for the system not to adapt at all.

We have also seen in previous sections that imperfect sensing is a source of uncertainty that can affect the normal operation of self-adaptation, for instance, by inducing an error in the observed value of the response time variable $\hat{r}$.

Figure 5 illustrates the interaction between imperfect sensing (in r) and uncertainty in tactic latency (in ActivateServer). The figure illustrates a situation in which response time goes above threshold $R_{\max}$ and triggers the tactic. One of the first things that we can observe is that the error in the sensing induces a delay in the triggering of the tactic, which would be triggered at time t_r^t without sensing error (r, dashed red line), and at $t_{\hat{r}}^t$ when based on the observed value of the response time ($\hat{r}$, dotted red line).

Beyond that delay in tactic execution, we can observe that there is an error induced in the measurement of the satisfaction of system goals. Let us recall that in Znn.com the satisfaction of goals is measured by means of accrued instantaneous utility (cf. Expression 1). Hence, we can characterize the overall error in measured accrued utility (corresponding to the performance concern u_p) induced by imperfect sensing in an arbitrary time interval $[t_1, t_2]$ as:

$$me_{u_p}(t_1, t_2) \equiv \int_{t_1}^{t_2} |u_p(r(t)) - u_p(\hat{r}(t))| \mathrm{d}t \tag{2}$$

For the overall execution of the system, we can say then that the overall accrued utility error due to measurement uncertainty is $me_{u_p}(0, T)$. In Fig. 5, this error corresponds (i.e., is proportional) to the light yellow and green areas between the values of the ground truth and observed variable values (dashed and dotted lines). Note that the areas in the figure are for illustration purposes and assume a simple linear mapping between observed response time values and utility. However, due to the arbitrary shape of utility functions (e.g., nonlinearities such as penalties associated with high response times), the error in measured utility might actually be larger or smaller, depending on the specific case.

Let us focus now on tactic latency. We assume that the effects of executing the tactic can take place in an interval of minimum/maximum latency $[t_{lmin}, t_{lmax}]$ and that the occurrence of this event in time is distributed according to a probability distribution (that we abstract away for clarity). Hence, if no sensing imperfection exists, the effect of the tactic execution will take place in the interval $[t_r^e = t_r^t + t_{lmin}, t_r'^e = t_r^t + t_{lmax}]$, whereas with the sensing imperfection, in this example the effect would take place in the interval $[t_{\hat{r}}^e = t_{\hat{r}}^t + t_{lmin}, t_{\hat{r}}'^e = t_{\hat{r}}^t + t_{lmax}]$. Given these two alternative scenarios for execution, the maximum difference interval that can exist between the time instants in which the effects of the latency takes place is given by $[t_r^e, t_{\hat{r}}'^e]$. The case in which the effects can take place at the earliest time instant t_r^e (minimum overall latency l_{min}) corresponds to perfect sensing and minimum tactic latency. On the other end, the effects of the tactic can take place at the latest (time instant $t_{\hat{r}}'^e$, maximum overall latency $l_{\max}$) when imperfect sensing is given along with maximum tactic latency.

This means, that during the period $[t_r^e, t_{\hat{r}}'^e]$, there are two different contributions by different uncertainties to the error in the measurement of accrued utility: one given by the measurement error of r that corresponds to $me_{u_p}(t_r^e, t_{\hat{r}}'^e)$ (green areas in the figure), and another one that corresponds to the alternative, delayed execution of the tactic (r').

The contribution to the error in measured accrued utility of these two combined sources can be characterized as follows:

$$lme_{u_p} \equiv \int_{t_r^e}^{t_{\hat{r}}'^e} |u_p(\hat{r}(t)) - u_p(\hat{r'}(t))| \mathrm{d}t \tag{3}$$

In Fig. 5, we illustrate the integrated error over time that corresponds to the combination of the green and pink areas of the figure, where we observe that the error resulting from the interaction of the two uncertainties is different to the sum of the different contributions considered individually. Of course, this example is just one instance that illustrates possible interactions that may occur between imperfect sensing and tactic latency: other situations are possible, such as anticipated triggering of the adaptation tactic due to measurement error, or effects in system variables that outlast the maximum latency period (which we chose to bound in our example).

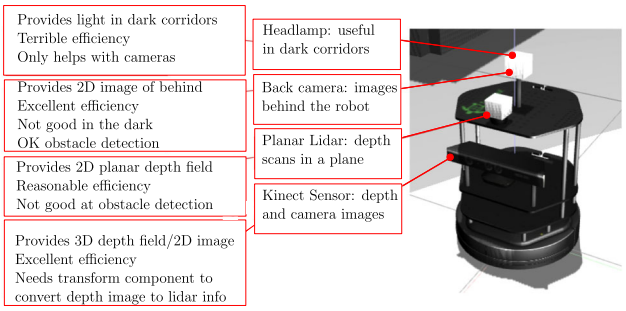

Fig. 6 Mobile robotics architecture configuration space

3.2 Autonomous mobile service robot

Mobile indoor service robots operate in environments where obstacles might dynamically appear, light conditions may change, and batteries may require recharging. They are also limited in what they can sense, creating uncertainty in their location, chances of colliding against obstacles, and the resources that they may have left to complete a plan. In spite of this uncertainty, they must attempt to ensure safe operation, effective use of resources like battery, and timeliness of completing a task.

In a simple scenario, the mission of the robot is navigating to a target location from an initial location in the shortest possible time, with a limited battery, and without bumping into obstacles or walls. To achieve this goal, the robot can perform physical actions (e.g., move between locations) and change its configuration (e.g., change a sensor, its localization algorithm, or its speed setting). While accomplishing the mission, the main goals are: (i) *timeliness*—the robot should get to the target in the shortest possible time, (ii) *safety*—the robot should arrive at the target location without bumping into obstacles, and (iii) *efficiency*—the robot should minimize the energy used to get to the target location (Fig. 6).

The MAPE-K architecture of the self-adaptation layer synthesizes specifications for the architecture and behavior of the robot to successfully complete the mission, attending to the criteria described above, despite situations that include component or sensor failure, obstacles blocking corridors, and unexpectedly low battery level.

3.2.1 Sources of uncertainty

Some of the uncertainties that can affect this robot system are listed below, classified according to their sources.

- *Model*: The fidelity of the models used to make decisions may cause errors in the system behavior. For example, an overly abstract model of the robot or its environment may cause collisions with external objects because some of their protruding parts have not been considered in the models, and hence clash with the environment obstacles. Similarly, a coarse-grained discretization of the navigation system or a very low resolution of the navigation time step can produce that the robot reacts too late to unexpected situations, e.g., not being able to stop in time when it detects an obstacle or another robot crosses its path.
- *Adaptation functions*: The outcome of executing a given adaptation tactic (e.g., changing the navigation component) is unknown. Sensing is also imperfect, so decisions based on the values of sensors' measurements (e.g., the position of the robot and surrounding obstacles based on information coming from the cameras) may be wrong, or at least carry some uncertainty. In some cases, information coming from sensors may be inaccurate due to, e.g., miscalibrated cameras, and even incomplete (obstacles that do not intersect with the plane of the lidar cannot be detected). The time that an adaptation tactic takes (i.e., its *latency*) is also subject to uncertainty, e.g., the time it takes to change the localization algorithm.
- *Goals*: Robot goals (timeliness, safety and efficiency) clearly conflict, so trade-offs between them need to be established. In addition, the weight assigned to each objective can change, and also be different depending on the stakeholders' opinions. Of course, stakeholders are not 100% confident about their opinions, so there is an associated uncertainty that needs to be handled. Besides, these opinions and their associated uncertainties need to be merged and reconciled in order to find consensus decisions.
- *Environment*: New obstacles can unexpectedly appear.
- *Resources*: The remaining amount of energy cannot be directly measured and has to be estimated based on the output voltage.
- *Managed system*: The robot may suffer erratic or intermittent failures in some of its components.

In this case, interactions between these uncertainties may also happen and cause undesirable effects. For example, a too conservative design that tries to avoid movements with too much degree of uncertainty in several of these sources (potential objects in front, low confidence in the readings of a cheap camera or low precision of sensors) may cause the robot to move too cautiously and slowly, and therefore fail to achieve its goals. Conversely, a more aggressive design that virtually ignores all uncertainties in order to achieve its goals at all costs may cause the robot to collide with other objects, suddenly run out of battery power because it stretches it to the limit, or make mistakes when grasping objects or performing its tasks. Again, it is crucial to explicitly represent and quantify the uncertainty associated with each source, decide the appropriate manner to combine these individual uncertainties when they interact, and compute the combined

uncertainty in order to make better informed decisions. We illustrate two of such interactions in the remainder of this section.

3.2.2 Uncertainties due to model and environment

Goals in cyber-physical systems tend to be of a different nature, when compared to IT systems like Znn.com and therefore can interact in different ways with other sources of uncertainty. Consider for instance the uncertainty associated with the abstraction of the models that capture the physical environment of the robot. An overly abstract model might cause computation of sub-optimal navigation paths or collisions against objects that would not be given with a more detailed model. This uncertainty affects all three nonfunctional goals of the robot, degrading safety due to the extra collisions, timeliness due to the time required to maneuver and recover from them, and efficiency due to all the energy consumed during the recovery.

In addition to the degraded nonfunctional goals, the unexpected appearance of obstacles that were not present in the environment model can compound with the uncertainty induced by model abstraction, producing further effects.

Figure 7 illustrates a scenario in which the robot has to traverse a corridor and arrive at a target location on the right, overcoming the obstacle and the person in the middle. While the obstacle is fixed and its presence known by the robot (i.e., it is captured in its model), the presence of the person is unknown a priori and considered part of the uncertainty associated with the future evolution of the environment. The model of the obstacle encoded in the robot is also considered in two variants: a high-resolution version that includes the geometry of its protrusions, and a coarse-grained variant consisting of an inaccurate bounding box. Figure 7a shows the situation in which there is no uncertainty associated with the model or the environment (i.e., the robot has an accurate portrayal of the geometry of the obstacle and knows about the presence of the person in the corridor). In this case, the robot goes around the obstacle through the gap without any problems. Figure 7b shows the situation in which the robot knows about the presence of the person, but is not aware of the details of the geometry of the obstacle. In this case, the robot again chooses to go through the gap, but collides with the obstacle before resuming its navigation towards the target location, incurring a penalty in safety, timeliness, and energy consumption. In Fig. 7c, the robot does not know about the presence of the person in the corridor, so it decides to go through that side of the obstacle, and unexpectedly finds that the corridor is blocked. This triggers replanning, which incurs a time and energy consumption penalty. Next, the robot decides to go around the obstacle through the other side and arrives at its destination. In Fig. 7d we observe the case in which the two sources of uncertainty are combined. The

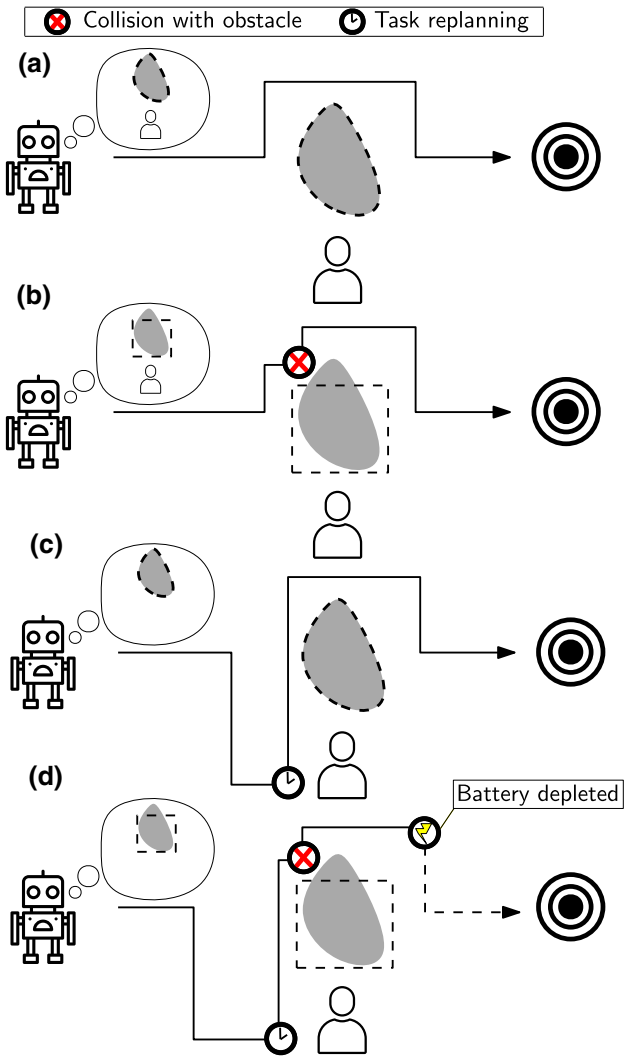

Fig. 7 Environment and model uncertainty interaction: **a** no uncertainty, **b** model uncertainty, **c** environment uncertainty, and **d** combined environment and model uncertainty

robot decides to go through the side of the corridor where the human is, needs to replan, and decides to go through the other side of the obstacle. However, in this case the inaccuracies in the model cause the robot to collide with the obstacle, which requires further time, maneuvering, and energy consumption. At this point, the interaction between the two types of uncertainty can make other effects emerge that go beyond the sum of the parts (i.e., degraded timeliness, energy consumption, and safety). For instance, the battery of the robot may not have enough energy to complete the mission due to the extra energy spent in the combination of the energy expense required for replanning and recovering from the collision. This situation would not be given in any of the situations in which the sources of uncertainty are given individually.

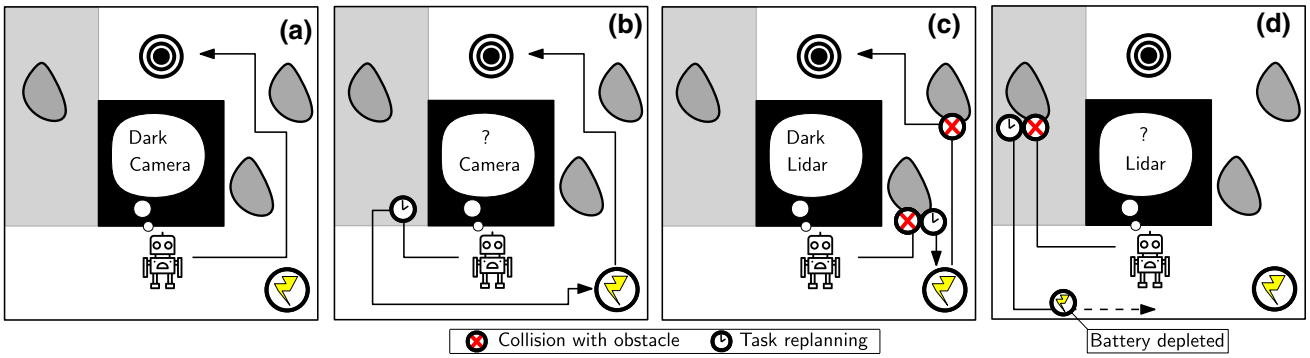

Fig. 8 Environment and sensing uncertainty interaction: **a** no uncertainty, **b** environment uncertainty, **c** sensing uncertainty, and **d** combined environment and sensing uncertainty

3.2.3 Uncertainties due to environment and adaptation functions

Environment uncertainty can also interact with adaptation functions such as sensing, and produce effects that only emerge when the two are combined.

Let us continue with our robot example, which is now in the scenario depicted in Fig. 8. In this scenario, the goal of the robot is arriving at the target location on top (depicted as concentric circles), avoiding obstacles, and with limited battery. One of the two corridors that the robot can use to reach the target location is dark (left, in gray). Environmental uncertainty in the scenario can occur when the robot is not aware that the left corridor is dark (represented by a question mark in the thought bubble). Sensing uncertainty is present when the robot is using a planar lidar sensor to detect obstacles, instead of a camera. In the example, we assume that the obstacles are not detectable via lidar because they do not intersect with the lidar plane and therefore the robot is completely unaware of their presence.

Scenario (a) illustrates the case in which there is no uncertainty: the robot is aware that the left corridor is dark and the camera is able to detect obstacles. Hence, the robot's planner generates a task plan to reach the target location through the right corridor and everything runs smoothly.

In scenario (b), the robot is not aware of the lack of light in the left corridor. Hence, the planner determines to go through it, but when the robot arrives at the corner, its sensors detect that the corridor is dark. At that point, task replanning is triggered. The planner generates a plan to go through the right corridor. However, due to the limited battery, the robot stops by the charging station in the corner before continuing the route towards the target location.

Scenario (c) depicts the scenario in which the robot knows that the left corridor is dark, but in this case the sensor built in to detect obstacles is the lidar instead of the camera. In this case, the robot's planner generates a task plan to go through the right corridor. However, due to the lack of obstacle detection capabilities, the robot collides against the first obstacle. At this point, the robot's energy analysis determines that the estimated level of battery might not be enough to complete the mission, so the task plan is regenerated to charge in the station, and then proceed towards the target location through the dark corridor again. The robot collides again with the second obstacle, but despite the safety penalty incurred due to the collisions, it manages to accomplish the mission.

Finally, scenario (d) illustrates the case in which environment and sensing uncertainty are combined: the robot does not know about the lack of light in the left corridor, and the built in sensor is not able to detect any obstacles. In this case, the robot's planner determines that the robot should go through the left corridor. Unlike in case (b), going through the dark corridor should not represent any problems a priori because, unlike the camera, the lidar sensor is not sensitive to low-light conditions. Hence, the lack of light in the corridor does not trigger any replanning, and the robot keeps on advancing through the dark corridor until it collides with the obstacle, which blocks the way and hence, the robot cannot progress. At this point, replanning is triggered, and the robot decides to go back to the charging station, and then proceed to the target location through the right corridor. However, in this case the battery is depleted before arriving at the charging station and the mission fails.

An interesting observation that we can make is that if the robot's planner had decided to go through the left corridor in the first place in scenario (c), the outcome would have been similar to that of scenario (d). This observation illustrates that the ways in which uncertainty from different sources interact can be subtle in many situations, and that the provision of solid guarantees about run-time system behavior in software intensive systems demands further study about uncertainty interaction.

4 Challenges

Based on the current limitations to mitigate uncertainty interaction in self-adaptive systems as in the examples discussed in Sect. 3, we have identified a set of challenges, which need to be addressed as self-adaptive systems become more prevalent, particularly in safety-critical sectors. We also classify them in different categories related to modeling, analysis, mitigation, and exploration.

4.1 Modeling challenges

These challenges refer to how the approach to modeling uncertainties can influence the mitigation of the uncertainty interaction problem. The following challenges have been identified:

Challenge M1: Combining uncertainties with different representations. Different types of uncertainties require disparate notations. For example, a lack of response from a sensor whose value is needed for deciding about a change of behavior can be due to a delayed transmission or to the fact that the sensor's battery is exhausted and therefore the sensor will never respond. How long to wait for the response? And, if it is received late, how much can we trust its value? In this case, we need to represent all the elements required to capture and quantify the combined uncertainty to decide whether the overall degree of uncertainty discourages any actions based on it. Similarly, think of the combination of a sensor whose measurements are expressed as fuzzy values, with a decision threshold that uses probabilistic logic to make decisions. Research is needed in the quest for notations and logics that enable the combination of uncertainties of different nature, or are specified using different notations.

Challenge M2: Combining uncertainties with different granularity, resolution, or abstraction levels. Even if the interacting uncertainties are expressed in similar notations and use similar reasoning mechanisms, they may be defined at different levels of abstraction or with different levels of granularity. One challenge is how to discover and compute the influence that each of them may have in the interaction? For instance, consider a moving robot that has a positioning device with a precision of 1 cm, and a move base with motors whose minimum energy pulses last 1 second and make the robot move a minimum of several cm, moving in an unknown environment. In this context, any decision made by the robot controlling software should be carefully considered because the coarse-grained granularity of the movements can introduce significant imprecision and vagueness—see the different uncertainties described in Sect. 2.3. There is a need for faithful abstraction and refinement mechanisms that allow us to balance the levels of abstraction and/or resolution of the corresponding uncertainties, while respecting the system properties of interest. For example, interpolation techniques can be used to approximate the more abstract model with a more refined one. Other mechanisms should also be explored, as well as how they preserve (or degrade) the properties of interest of the system.

Challenge M3: Combinatorial explosion of uncertainty interaction effects. Although we managed to combine different interacting uncertainties, the effects of the combination need to be quantified and bounds established. The challenge here lies in how to analyze and measure the effects of such a combination, which may be of an exponentially amplifying character. For example, a variability model describing the possible configuration options under uncertain environmental requirements may be significantly worsened by imprecise values of the variables used to determine the option to choose. Methods and techniques for coping with these situations are needed. Furthermore, a better understanding of how two or more uncertainties interact can also help to define limits to the effects of their combination. Most models of uncertainty specification and analysis follow the worst-case scenario. However, in the physical world we see how uncertainties cancel each other out or at least offset each other's effects. Defining alternative models to those using worst-case analysis (e.g., using means, medians or other central values measures) that are more faithful to the way in which real-world systems is a promising direction to overcome this challenge.

4.2 Analysis challenges

In order to be able to manage uncertainty interactions in self-adaptive systems, we need to be able to identify such interactions, quantify their impact, and determine those that require mitigation, leading to the following challenges:

Challenge A1: Identifying uncertainty interactions. Determining all the uncertainty interactions that a self-adaptive system needs to consider is extremely difficult. While Sect. 3 provides multiple examples of such interactions for two prototypical self-adaptive systems, assembling a comprehensive list of these interactions for a given system is a complex and error-prone process. Methods adapted from risk identification could potentially be used for this purpose, supported by predefined lists of likely uncertainty interactions and contributions from both domain experts and the developers of the actual system.

Challenge A2: Quantifying the impact of uncertainty interactions. Assuming that a complete list of relevant uncertainty interactions could be compiled, the next challenge is to determine their potentially disparate impacts. The interaction between the model uncertainty due to a coarse-grained discretization of the map used for robot navigation and sensing uncertainty that makes obstacle detection imprecise can have a considerable impact on a robot's ability to navigate through an environment containing obstacles. In contrast, the

interaction between the same type of model uncertainty and effecting uncertainty due to actuator imprecision may have only a limited impact if the system goals allow such imprecision.

Challenge A3: Determining the uncertainty interactions that require mitigation. The effort to devise, implement, test and deploy suitable mitigations for uncertainty interactions can be considerable. As such, methods are required for systematically determining which of these interactions need to be addressed, and which can be accepted. These methods must consider the impact of all relevant uncertainty interactions, and must carry out this assessment based on a set of well-defined criteria provided by domain experts. Where applicable, the intended users of the system may need to be involved in this assessment. For instance, uncertainty interactions that impact the navigation of an assistive-care robot need to be mitigated when the robot is helping a partially sighted user, but may be acceptable for a fully sighted user.

4.3 Mitigation challenges

When dealing with uncertainty in self-adaptive systems, one promising approach is to explicitly and proactively mitigate uncertainty through uncertainty-reduction techniques [7,31]. The key idea is to allocate system resources to reducing uncertainty in contexts where inaccurate decisions might have a strong negative impact on system utility. For example, in a robotic navigation scenario, a robot might decide to turn on a spotlight in a darkened hallway to reduce uncertainty in robot localization. Such actions, however, come with a cost (e.g., in additional energy consumed, more intrusive presence, etc.) and hence it becomes important to reason about the net effect of such uncertainty reduction techniques on overall utility. When considering interactions between different forms of uncertainty, however, a number of challenges arise:

Challenge Mt1: Uncertainty mitigation dominance.

Uncertainty reduction in one dimension may dominate, and possibly make irrelevant, uncertainty mitigation in other dimensions. Consider the robot example. One concern for such systems is energy usage—we do not want the robot to run out of battery power en route to its destination. Under normal operating conditions we may choose to query the power level to reduce uncertainty about its usage. But such monitoring may be completely dominated by the need to reduce uncertainty in the environment, for example by turning on the robot's headlamp in a poorly lighted space.

Challenge Mt2: Uncertainty mitigation augmentation. A second form of uncertainty mitigation interaction is augmentation: by reducing the uncertainty in one dimension we may also reduce uncertainty in other dimensions. For a robot, lighting a hallway to reduce localization uncertainty may also affect occupancy uncertainty (knowing how many people are in the space, and hence how intrusive the robot is). In such a situation, it might be wise to pick an uncertainty mitigation approach that is less effective in one dimension, but through augmentation can reduce uncertainty in multiple dimensions.

Challenge Mt3: Uncertainty mitigation conflicts. This challenge is related to the third form of uncertainty mitigation interaction, which is conflict: by reducing uncertainty in one dimension you may increase it in another. For a robot, turning on a headlamp may reduce localization uncertainty, but increase uncertainty about the robot's power level if the headlamp's energy consumption is not well-calibrated. It is important to explore mechanisms in order to reduce the uncertainty increase.

4.4 Exploration challenges

Several factors for current and emerging systems will necessitate sophisticated strategies for assessing the impact of uncertainty. Cost as defined in terms of human lives and monetary expense provides perhaps the most compelling motivation for run-time relevant impacts of uncertainty. While much of the work with uncertainty management centers around system development (e.g., how to make systems more robust and resilient to uncertainty), it is also important to understand the scope and impact of uncertainty. Specifically, techniques are needed to *explore* uncertainty and its impact. We highlight the following specific challenges:

Challenge E1: Complementary uncertainty exploration. Given the range of sources of uncertainty, varying types and degrees of impact, temporal relevance, and potentially conflicting strategies for mitigating uncertainty, complementary uncertainty exploration techniques are needed. For example, multi-objective optimization techniques can be leveraged to explore the cumulative impact of multiple sources of uncertainty. Probabilistic analysis, data mining, and (adversarial) machine learning techniques can be used to explore uncertainty based on historic data. For the ZNN application, historic-use data and uncertainty factors can be used to guide the self-adaptation for changes to the server configuration and networking support. Search-based techniques such as evolutionary computing can be used to explore uncertainty that is not predicated on previously known uncertainty data [10]. The different sources of uncertainty for the robot (e.g., terrain, lighting, obstacle size, wheel slippage) can all affect the robot navigation and obstacle avoidance. Evolutionary computing can be used to explore the different combinations of the uncertainty factors to determine contexts that would be detrimental to the robot behaving acceptably [27,28]. Exploring uncertainty with "What if?" scenarios (e.g., using game theory) enables the developer to explore uncertainty with respect to specific operational contexts [6,26].

Challenge E2: Utilization of digital twin frameworks. Digital twin (DT) frameworks provide a potentially invaluable

framework for uncertainty exploration that supports "human in the loop." Digital twinning can be used in a number of scenarios. For example, with uncrewed space missions, DTs can incorporate run-time monitored information regarding the environment and its uncertainty factors, which can be analyzed and explored in order to determine appropriate behavior changes for the onboard control behavior for a terrestrial rover. DT frameworks can also be used to explore "What if?" scenarios based on historic data and synthetic data [14,15]. Effective use of DTs for uncertainty exploration will necessarily have to be informed by advances across all the aforementioned challenge categories related to the representation, analysis, and mitigation of uncertainty interactions.

5 Conclusions

In this SoSym Expert Voice, we have described the *Uncertainty Interaction Problem* in self-adaptive systems, focusing on uncertainty modeling in an integrated fashion. The motivation has been illustrated with examples in two representative application domains (an autoscaling news website infrastructure and a mobile autonomous service robot). We have outlined a set of challenges that concern the representation, analysis, mitigation, and exploration of interactions among uncertainties from different sources. The set of potential uncertainty interactions illustrated in this article is by no means exhaustive and further collaborative effort from the self-adaptive systems and modeling communities will be required to develop a detailed catalog of uncertainty interactions and guidelines to deal with them. However, through the *Uncertainty Interaction Problem*, we hope to set the reference coordinates to reason about the emergent effects of uncertainty interactions and their impact on software-intensive systems. We believe that this is an important area that deserves the attention of the community, and that research in this direction will pave the way towards more holistic approaches that enable the construction of safer and more resilient software-intensive systems.

Acknowledgements This work was partially supported by the Assuring Autonomy International Programme project "Ambient Assisted Living for Long-term Monitoring and Interaction Integration," the European Commission (FEDER) and Junta de Andalucia under projects APOLO (US-1264651), MBT-I4A (P20-00067-FR) and EKIPMENT-PLUS (P18-FR-2895), by the Spanish Government (FEDER/Ministerio de Ciencia e Innovación—Agencia Estatal de Investigación) under project COSCA (PGC2018-094905-B-I00), by Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) through the Carnegie Mellon Portugal Program under Grant SFRH/BD/150643/2020, projects (POCI-01-0247-FEDER-045915, POCI-01-0247-FEDER-045907), by NASA (Award 80NSSC20K1720). Bencomo's work has been sponsored by the EPSRC Research Project Twenty20Insight (Grant No. EP/T017627/1). Cheng's work has been sponsored by National Science Foundation (DBI-0939454), Ford Motor Company, General Motors Research, and ZF; and the research has also been sponsored by Air Force Research Laboratory (AFRL) under agreement numbers FA8750-16-2-0284 and FA8750-19-2-0002. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copy-right 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 Air Force Research Laboratory (AFRL), the U.S. Government, National Science Foundation, Ford, GM, ZF, or other research sponsors.

References

1. Baresi, L., Pasquale, L., Spoletini, P.: Fuzzy goals for requirements-driven adaptation. In: 2010 18th IEEE International Requirements Engineering Conference, IEEE, pp. 125–134 (2010)
2. Bencomo, N.: Quantun: quantification of uncertainty for the reassessment of requirements. pp. 236–240 (2015). https://doi.org/10.1109/RE.2015.7320429
3. Bencomo, N., Belaggoun, A., Issarny, V.: Dynamic decision networks for decision-making in self-adaptive systems: a case study. In: 2013 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), pp. 113–122 (2013). https://doi.org/10.1109/SEAMS.2013.6595498
4. Bertoa, M.F., Burgueño, L., Moreno, N., et al.: Incorporating measurement uncertainty into OCL/UML primitive datatypes. Softw. Syst. Model. **19**(5), 1163–1189 (2020)
5. Burgueño, L., Muñoz, P., Clarisó, R., et al.: Dealing with belief uncertainty in domain models. ACM Trans. Softw. Eng. Methodol. (TOSEM) In submission (2022)
6. Cámara, J., Garlan, D., Moreno, G.A., et al.: Analyzing self-adaptation via model checking of stochastic games. In: de Lemos R, Garlan D, Ghezzi C, et al (eds) Software Engineering for Self-Adaptive Systems III. Assurances - International Seminar, Dagstuhl Castle, Germany, December 15–19, 2013, Revised Selected and Invited Papers, Lecture Notes in Computer Science, vol **9640**. Springer, pp. 154–187 (2013)
7. Cámara, J., Peng, W., Garlan, D., et al.: Reasoning about sensing uncertainty and its reduction in decision-making for self-adaptation. Sci. Comput. Program. **167**, 51–69 (2018)
8. Cheng, B., Sawyer, P., Bencomo, N. et al.: A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In: Proceedings of MODELS'09, pp. 468–483 (2009)
9. Cheng, B.H.C., de Lemos, R., Giese, H., et al.: Software engineering for self-adaptive systems: a research roadmap. In: Software Engineering for Self-Adaptive Systems, LNCS, vol **5525**. Springer, pp. 1–26 (2009b)
10. Cheng, B.H.C., Ramirez, A.J., McKinley, P.K.: Harnessing evolutionary computation to enable dynamically adaptive systems to manage uncertainty. In: 1st International Workshop on Combining Modelling and Search-Based Software Engineering, CMSBSE@ICSE 2013 (2013)
11. Cheng, S.W., Garlan, D.: Handling uncertainty in autonomic systems. In: In Proceedings of IWLU@ASE'07. ACM (2007). http://acme.able.cs.cmu.edu/pubs/uploads/pdf/IWLU07-HandlingUncertainties-pub.pdf

12. Critch, A.: (Retrieved 15 January 2019) Credence—using subjective probabilities to express belief strengths. http://acritch.com/credence/
13. DeMarco, T.: Controlling Software Projects: Management, Measurement & Estimation. Yourdon Press, New York (1982)
14. DeVries, B., Cheng, B.H.C.: Run-time monitoring of self-adaptive systems to detect n-way feature interactions and their causes. In: Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2018. ACM, pp. 94–100 (2018)
15. DeVries, B., Fredericks, E.M., Cheng, B.H.C.: Analysis and monitoring of cyber-physical systems via environmental domain knowledge & modeling. In: 16th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2021 (2021)
16. Esfahani, N., Malek, S.: Uncertainty in self-adaptive software systems. In: Software Engineering for Self-Adaptive Systems II, LNCS, vol **7475**. Springer, pp. 214–238 (2013)
17. Famelis, M., Chechik, M.: Managing design-time uncertainty. Softw. Syst. Model. **18**(2), 1249–1284 (2019)
18. Feller, W.: An Introduction to Probability Theory and Its Applications. Wiley, Hoboken (2008)
19. Giese, H., Bencomo, N., Pasquale, L., et al.: Living with Uncertainty in the Age of Runtime Models. In: Models@run.time, LNCS, vol **8378**. Springer, pp. 47–100 (2014)
20. Hao, J., Jiang, T., Wang, W., et al.: An empirical analysis of VM startup times in public IAAS clouds. In: 2021 IEEE 14th International Conference on Cloud Computing (CLOUD), pp. 398–403 (2021)
21. Hezavehi, S.M., Weyns, D., Avgeriou, P., et al.: Uncertainty in self-adaptive systems: a research community perspective. ACM Trans. Auton. Adapt. Syst. **15**(4) (2021)
22. Huebscher, M.C., McCann, J.A.: A survey of autonomic computing—degrees, models, and applications. ACM Comput. Surv. **40**(3) (2008)
23. JCGM 100:2008 (2008) Evaluation of measurement data—guide to the expression of uncertainty in measurement (GUM). http://www.bipm.org/utils/common/documents/jcgm/JCGM_100_2008_E.pdf
24. Jøsang, A.: Subjective Logic—A Formalism for Reasoning Under Uncertainty. Artif. Intell. Found. Theory Algorithms (2016)
25. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36** (2003)
26. Kinneer, C., Garlan, D., Goues, C.L.: Information reuse and stochastic search: managing uncertainty in self-* systems. ACM Trans. Auton. Adapt. Syst. **15**(1), 3:1-3:36 (2021)
27. Langford, M.A., Cheng, B.H.C.: Enhancing learning-enabled software systems to address environmental uncertainty. In: 2019 Conference on Autonomic Computing, ICAC 2019 (2019)
28. Langford, M.A., Cheng, B.H.C.: Enki: a diversity-driven approach to test and train robust learning-enabled systems. ACM Trans. Auton. Adapt. Syst. **15**(2), 5:1-5:32 (2021)
29. Mahdavi-Hezavehi, S., Avgeriou, P., Weyns, D.: A classification framework of uncertainty in architecture-based self-adaptive systems with multiple quality requirements (2017)
30. Moreno, G.A., Cámara, J., Garlan, D., et al.: Proactive self-adaptation under uncertainty: a probabilistic model checking approach. Assoc. Comput. Mach. **2015**, 1–12 (2015)
31. Moreno, G.A., Cámara, J., Garlan, D., et al.: Uncertainty reduction in self-adaptive systems. In: Andersson, J., Weyns, D. (eds) Proceedings of SEAMS@ICSE'18. ACM, pp. 51–57 (2018)
32. Oberkampf, W.L., DeLand, S.M., Rutherford, B.M., et al.: Error and uncertainty in modeling and simulation. Reliab. Eng. Syst. Saf. **75**(3), 333–357 (2002)
33. Oquendo, F.: Coping with uncertainty in systems-of-systems architecture modeling on the IoT with SosADL. In: Proceedings of SoSE'19, pp. 131–136 (2019)
34. Paterson, C., Calinescu, R.: Observation-enhanced QoS analysis of component-based systems. IEEE Trans. Softw. Eng. **46**(5), 526–548 (2020)
35. Perez-Palacin, D., Mirandola, R.: Uncertainties in the modeling of self-adaptive systems: a taxonomy and an example of availability evaluation. In: Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering, ICPE '14 (2014)
36. Ramirez, A.J., Jensen, A.C., Cheng, B.H.C.: A taxonomy of uncertainty for dynamically adaptive systems. In: 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2012. IEEE Computer Society (2012)
37. Rausand, M.: Risk Assessment: Theory, Methods, and Applications. Wiley, Hoboken (2013)
38. Russell, S., Chen, F.: Controlling AI. Podcast (2020)
39. Russell, S.J., Norvig, P.: Artificial Intelligence, A Modern Approach, 3rd edn. Prentice Hall, New Jersey (2010)
40. Samin, H., Bencomo, N., Sawyer, P.: Decision-making under uncertainty: be aware of your priorities. Softw. Syst. Model. (2022)
41. Schmerl, B.R., Cámara, J., Gennari, J., et al.: Architecture-based self-protection: composing and reasoning about denial-of-service mitigations. In: Proceedings of HotSoS'14. ACM, p. 2 (2014)
42. Seely, A.J., Macklem, P.T.: Complex systems and the technology of variability analysis. Crit. Care **8**, 367–384 (2004)
43. Thunnissen, D.P.: Uncertainty classification for the design and development of complex systems. In: Proceedings of the 3rd Annual Predictive Methods Conference, Veros Software (2003)
44. Troya, J., Moreno, N., Bertoa, M.F., et al.: Uncertainty representation in software models: a survey. Softw. Syst. Model. **20**(4) (2021)
45. Whittle, J., Sawyer, P., Bencomo, N., et al.: RELAX: incorporating Uncertainty into the specification of self-adaptive systems. In: Proceedings of RE'09, pp. 79–88 (2009)
46. Whittle, J., Sawyer, P., Bencomo, N., et al.: RELAX: incorporating uncertainty into the specification of self-adaptive systems. In: Proceedings of RE'09. IEEE Computer Society, pp. 79–88 (2009)
47. Zhang, M., Ali, S., Yue, T., et al.: Uncertainty-wise cyber-physical system test modeling. Softw. Syst. Model. **18**(2), 1379–1418 (2019)
48. Zimmermann, H.J.: Fuzzy Set Theory and Its Applications. Springer, Berlin (2001)

Javier Cámara is Associate Professor of Computer Science at the University of Málaga and Honorary Visiting Fellow at the Department of Computer Science, University of York. His current research interests include self-adaptive and autonomous systems, software architecture, formal methods, as well as cyber-physical and AI systems. During 2018–2021, he was a Lecturer in Computer Science at the University of York. Prior to that, he was part of the core faculty of the Institute for Software Research at Carnegie Mellon University (2015–2018). He received his European PhD with honors from the University of Máálaga in 2009. He has also been a postdoctoral research associate at INRIA Rhône-Alpes, the Centre for Informatics and Systems of the University of Coimbra, and Carnegie Mellon University. Contact him at jcamara@uma.es or visit http://www.javicamara.com/.

Javier Troya is Associate Professor of Software Engineering at the University of Málaga, Spain. Before, he was a post-doctoral researcher as well as Assistant and Associate Professor at the University of Seville, Spain (2016–2021), and a post-doctoral researcher in the TU Wien, Austria (2013–2015). He obtained his International PhD with honors from the University of Málaga, Spain (2013), and was awarded in 2020 with the I3 Certificate by the Spanish Ministry of Science, Innovation and Universities. His current research interests include model transformation testing, uncertainty modeling and digital twins. Contact him at jtroya@uma.es or visit http://webpersonal.uma.es/.

Antonio Vallecillo is Professor of Software Engineering at the University of Málaga, Spain, where he leads the Atenea Research Group. His current research interests include Model-based Software Engineering, Open Distributed Processing, and Software Quality. Information about his publications, research projects and activities can be found at http://www.lcc.uma.es/~av.

Nelly Bencomo is Associate Professor at Durham University in the UK. Her research interests in Software Engineering include Decision-making under Uncertainty, Model-Driven Engineering, Autonomous Systems, Artificial Intelligence and Requirements Engineering. She was awarded EU MC and UK Leverhulme Individual Fellow ships. She is the PI of the EPSRC Project Twenty20 Insight. Contact her at nelly@acml.org or visit https://www.nellybencomo.me/.

Radu Calinescu is Professor of Computer Science at the University of York (UK). His main research interests are in formal methods for self-adaptive, autonomous, secure and dependable software, cyber-physical and AI systems, and in performance and reliability software engineering. He is an active promoter of formal methods at runtime as a way to improve the integrity and predictability of self-adaptive and autonomous systems and processes.

Betty H. C. Cheng is Professor in the Department of Computer Science and Engineering at Michigan State University. She is also the Industrial Relations Manager and senior researcher for BEACON, the National Science Foundation Science and Technology Center in the area of Evolution in Action. Her research interests include self-adaptive autonomous systems, requirements engineering, model-driven engineering, automated software engineering, and harnessing evolutionary computation and search-based techniques to address software engineering problems. These research areas are used to support the development of high-assurance adaptive systems that must continuously deliver acceptable behavior, even in the face of environmental and system uncertainty. Example applications include intelligent transportation and vehicle systems. She collaborates extensively with industrial partners in her research projects in order to ensure real-world relevance of her research and to facilitate technology exchange between academia and industry. Her collaborators include Ford, General Motors, ZF, BAE, Motorola, and Siemens. Previously, she was awarded a NASA/JPL Faculty Fellowship to investigate the use of new software engineering techniques for a portion of the NASA space shuttle software. She has recently launched new projects in the areas of model-driven approaches to sustainability, cyber security for automotive systems, and feature interaction detection and mitigation for autonomic systems, all in the context of operating under uncertainty while maintaining assurance objectives. Her research has been funded by several federal funding agencies, including NSF, AFRL, ONR, DARPA, NASA, ARO, and numerous industrial organizations. She serves on the journal editorial boards for Requirements Engineering and Software and Systems Modeling; she is Co-Associate Editor-in-Chief for IEEE Transactions for Software Engineering, where she previously served twice as an Associate Editor. She was the Technical Program Co-Chair for IEEE International Conference on Software Engineering (ICSE-2013), the premier and flagship conference for software engineering. She received her Bachelor of Science degree from Northwestern University, and her MS and Ph.D. from the University of Illinois-Urbana Champaign, all in computer science. She may be reached at the Department of Computer Science and Engineering, Michigan State University, 3115 Engineering Building, 428 S. Shaw Lane, East Lansing, MI 48824; chengb@msu.edu; www.cse.msu.edu/~chengb.

David Garlan is Professor of Computer Science and Associate Dean in the School of Computer Science at Carnegie Mellon University. His research interests include software architecture, selfadaptive and autonomous systems, formal methods, and cyber-physical systems. He is recognized as one of the founders of the field of software architecture, and in particular, formal representation and analysis of architectural designs. He has received a Stevens Award Citation for "fundamental contributions to the development and understanding of software architecture as a discipline in software engineering," an Outstanding Research award from ACM SIGSOFT for "significant and lasting software engineering research contributions through the development and promotion of software architecture," an Allen Newell Award for Research Excellence, an IEEE TCSE Distinguished Education Award, and a Nancy Mead Award for Excellence in Software Engineering Education. He is a Fellow of the IEEE and ACM.

Bradley Schmerl is a Principal Systems Scientist in the School of Computer Science at Carnegie Mellon University. His research interests include languages, tools, and frameworks to support software architecture, self-adaptive systems, and robotics software. He received an Allen Newell Award for Research Excellence for contributions made to software architecture. He is a Senior Member of the IEEE and ACM.