Formal Synthesis of Uncertainty Reduction Controllers

Marc Carwehl
carwehl@cs.hu-berlin.de
Institut für Informatik, Humboldt-Universität zu Berlin
Berlin, Germany

Calum Imrie
calum.imrie@york.ac.uk
Department of Computer Science, University of York
York, UK

Thomas Vogel
thomas.vogel@cs.hu-berlin.de
Institut für Informatik, Humboldt-Universität zu Berlin
Berlin, Germany

Genaína Rodrigues
genaina@unb.br
Department of Computer Science, University of Brasilia
Brasilia, Brazil

Radu Calinescu
radu.calinescu@york.ac.uk
Department of Computer Science, University of York
York, UK

Lars Grunske
grunske@cs.hu-berlin.de
Institut für Informatik, Humboldt-Universität zu Berlin
Berlin, Germany

ABSTRACT

In its quest for approaches to taming uncertainty in self-adaptive systems (SAS), the research community has largely focused on solutions that adapt the SAS architecture or behaviour in response to uncertainty. By comparison, solutions that reduce the uncertainty affecting SAS (other than through the blanket monitoring of their components and environment) remain underexplored. Our paper proposes a more nuanced, adaptive approach to SAS uncertainty reduction. To that end, we introduce a SAS architecture comprising an *uncertainty reduction controller* that drives the adaptive acquisition of new information within the SAS adaptation loop, and a tool-supported method that uses probabilistic model checking to synthesise such controllers. The controllers generated by our method deliver optimal trade-offs between SAS uncertainty reduction benefits and new information acquisition costs. We illustrate the use and evaluate the effectiveness of our approach for mobile robot navigation and server infrastructure management SAS.

CCS CONCEPTS

• **Computer systems organization** → **Reliability**; **Robotic autonomy**; • **Software and its engineering** → **Layered systems**; *Model-driven software engineering*; **Formal methods**.

KEYWORDS

controller synthesis, uncertainty, self-adaptive systems

ACM Reference Format:
Marc Carwehl, Calum Imrie, Thomas Vogel, Genaína Rodrigues, Radu Calinescu, and Lars Grunske. 2024. Formal Synthesis of Uncertainty Reduction Controllers. In *19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '24), April 15–16, 2024, Lisbon, AA, Portugal.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3643915.3644095

SEAMS '24, April 15–16, 2024, Lisbon, AA, Portugal

ACM ISBN 979-8-4007-0585-4/24/04.
https://doi.org/10.1145/3643915.3644095

1 INTRODUCTION

Essential services from all sectors of the economy and society rely on the effective operation of complex software-intensive systems. These systems range from sophisticated road traffic management software and public clouds running business-critical applications to cyber-physical systems from manufacturing. More often than not, they are used in real-world applications characterised by high levels of uncertainty related, for instance, to environmental changes, component failures, measuring inaccuracies, and user actions. To deliver their required functionality in such circumstances, software-intensive systems need to "tame" this uncertainty through *self-adaptation* [8, 21]. Self-adaptation is a process that involves the use of closed-control loops to *monitor* the system and its environment for relevant changes, to *analyse* the impact of these changes, to *plan* system adaptations that accommodate the changes, and to *execute* (i.e., to implement) these adaptations. Software-intensive systems that employ such monitor-analyse-plan-execute (or 'MAPE', cf. [14]) control loops are termed *self-adaptive systems* (SAS).

The growing demand for SAS in many application domains [32, 33] has led to the development of numerous self-adaptation solutions over the past two decades. Nevertheless, the vast majority of these solutions focus on determining and performing SAS adaptation tactics that take uncertainty into account. The complementary approach of *reducing epistemic uncertainty* (i.e., the uncertainty due to insufficient knowledge)—other than through a blanket monitoring of the system and its environment in the first step of the MAPE loop—is largely unexplored by existing SAS solutions.

In this paper, we argue that SAS can achieve better tradeoffs between adaptation outcomes and costs by combining established uncertainty-aware adaptation solutions with an adaptive approach to epistemic uncertainty reduction. To motivate the need for our hybrid self-adaptation paradigm, we refer to SAS exemplars proposed by the SEAMS research community [7].

As a first example, consider the UNDERSEA exemplar from [9]. This is an underwater robot tasked with measuring the oceanic salinity or temperature with a given accuracy and under energy use constraints, which requires the robot to adaptively switch on/off its sensors and to vary its speed depending on environmental conditions. Assuming that the mission needs to be performed within a designated perimeter which contains obstacles such as underwater rocks, the MAPE loop controlling the robot needs to consider the

robot's position in its decision-making. However, because of variable underwater currents that are difficult to model, this position is affected by epistemic uncertainty whose resolution requires the robot to navigate to the sea surface for GPS access. This uncertainty-reduction operation consumes significant time and energy. As such, it should ideally be performed adaptively (based on need), by considering factors such as the estimated distance between the robot and the nearest obstacle/perimeter boundary, robot speed, and estimated underwater current direction and speed.

As another example, consider the TAS exemplar from [30]. TAS is a telehealth service-based system that uses third-party services (i) to analyse the vital parameters of home-based patients with long-term health conditions, and, when the patient condition changes significantly, (ii) to order new medication, or (iii) to alert a medical team. Its MAPE loop is responsible for ensuring that the TAS reliability and response time remain within acceptable bounds, by switching to a functionally equivalent "backup" service when the reliability or performance of any of its three services becomes inadequate. However, because the backup services can also experience technical difficulties, the reliability and response time that the MAPE loop assumes for each of them are affected by epistemic uncertainty. To reduce this uncertainty, and thus to avoid switching to a backup service that has become unavailable or too slow, TAS should occasionally test these services by invoking their functions. Given the unavoidable overheads of these uncertainty-reduction invocations, their timing, frequency and number should be continually adapted to reflect the current TAS workload, the recent reliability and performance trends of the services used by TAS, etc.

Problem definition: The UNDERSEA and TAS scenarios we described are instances of a general problem faced by SAS whose MAPE loops make decisions based on estimates of variables affected by epistemic uncertainty. The precision of these estimates can be increased by using SAS-specific uncertainty-reduction "services",[1] accessing these services incurs a cost that may consist of CPU or memory overheads, bandwidth or energy use, carbon footprint, etc. As such, using these services continuously is unaffordable. Moreover, invoking them with a constant frequency is likely to yield suboptimal tradeoffs between the uncertainty-reduction cost and the effectiveness of the MAPE decision-making. Thus, the *adaptive uncertainty reduction problem* tackled in our paper is to determine *which uncertainty-reduction service(s) should be invoked when* in order to ensure that the SAS goals are optimally satisfied.

Our approach: To address the problem summarised above, we introduce a new paradigm for adaptive uncertainty reduction in self-adaptive systems, or Parley[2] for short. The fundamental premise behind Parley is that acquiring new knowledge to reduce epistemic uncertainty represents one of the paramount tasks that a SAS should be concerned with. In line with this premise, Parley is underpinned by a new SAS architecture (Fig. 1) that includes a dedicated *uncertainty reduction controller* (URC). The role of this new type of controller is to monitor the estimated values of uncertainty-affected variables that the existing MAPE loop (referred to here as *uncertainty-aware controller (UAC)*) operates with and to dynamically adapt this controller's use of the available uncertainty-reduction services by considering factors such as those from our earlier UNDERSEA and TAS scenarios. The use of different controllers for uncertainty reduction and managed-system adaptation to uncertainty in our Parley architecture has two major benefits. First, as with any architecture that promotes a separation of concerns between different system functions, Parley can lead to less complex and easier to maintain control loops than a monolithic architecture. Second, our two-tier control architecture makes the augmentation of an existing SAS with an uncertainty reduction controller straightforward.

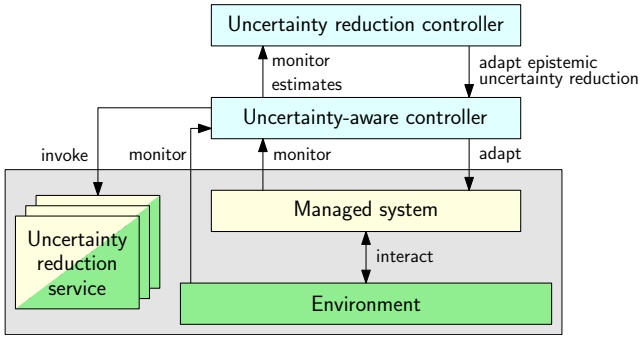


Figure 1: Parley self-adaptive system architecture: an uncertainty reduction controller drives the adaptive reduction of epistemic uncertainty through the invocation of uncertainty reduction "services" provided by the managed system, its environment, or a combination thereof.

In addition to this new architecture, Parley comes with a tool-supported method for the formal synthesis of uncertainty reduction controllers for SAS whose behaviour can be modelled using probabilistic state-transition models such as discrete-time Markov chains (DTMCs). This method uses a combination of probabilistic model checking and multi-objective genetic algorithms to synthesise URCs guaranteed to satisfy strict reliability, performance and other quality constraints, and to be Pareto-optimal with respect to a set of quality optimisation objectives.

Contributions: The main contributions of our paper are:

- The Parley hybrid self-adaptation paradigm, and associated two-tier controller architecture;
- The Parley method for synthesising correct-by-construction uncertainty reduction controllers;
- A toolchain which automates the application of the URC synthesis method, and which includes a new tool for augmenting the discrete-time Markov chain model of a SAS with the new states and transitions required for the URC synthesis;
- The evaluation of Parley within SAS case studies from the mobile robot navigation and server infrastructure management domains.

Paper structure: The rest of the paper is organised as follows. In Section 2, we compare Parley to related research on self-adaptive systems. Next, Section 3 provides a running example used to illustrate the components and stages of Parley in its description

[1] These services may suffer from *aleatoric uncertainty*, i.e., irreducible uncertainty due, for instance, to natural variability; e.g., the GPS services used by the UNDERSEA robot from our example can only provide the robot location with a certain accuracy.
[2] From the French *parler* (to speak); parley means to discuss and come to an agreement.

from Section 4. Finally, Section 5 presents our evaluation of Parley, and Section 6 concludes the paper with a brief summary and suggestions for further work.

2 RELATED WORK

Handling uncertainty has been one of the driving forces of research in the area of self-adaptive systems [4, 12, 21, 26, 31, 35]. In the following, we review selected approaches that are particularly related to Parley. Uncertainty handling may follow a control-theoretical [15, 23, 28, 29], an architecture-based [5, 22, 24, 25, 34] or a planing-based [1, 3, 8, 22, 25, 27] adaptation approach.

Control-theoretical approaches: Shevtsov et al. [28] apply control theory to deal with adaptation problems for systems with strict goals and control theoretical requirements (set point, thresholds, optimisation) as well as to handle and provide assurances under various sources of uncertainty. In contrast, Michelmore et al. [23] developed a framework for evaluating the safety of autonomous driving using end-to-end Bayesian Neural Network (BNN) controllers. With their work, uncertainty estimates for the controller's decisions can be obtained with a priori statistical guarantees. While in their work they optimise for safety constraints, our optimal policy aims for a broader spectrum where epistemic uncertainties can be mitigated as the system objectives are satisfied. The work by Kobayashi et al. [15] proposes a methodology to build more robust controllers against perceptual uncertainty through an automated workflow. By providing optimal policies over the behaviour, our work goes one step further as it takes probabilities into account and provides means to build extensible controllers through the separation of concerns between uncertainty-aware and -reduction controllers. Moreover, the controller synthesis in Parley can inherently accommodate not only multiple parameters but also multiple thresholds through trading off objectives. Solano et al. [29] propose an assurance process to handle uncertainty through a generative approach that uses a goal model augmented with uncertainties. As an outcome, reliability and cost algebraic formulae are used by a PID controller to provide policies and assure the properties of the managed system. Their solution, however, does not allow for architecturally decoupling the controllers' behaviours and concerns, which could render the controller's maintainability and scalability infeasible.

Architecture-based adaptation approaches: Regarding the handling of uncertainties following an architecture-based adaptation approach, Weyns and Iftikar [34] propose ActivForms-ta, an architecture-based adaptation approach based on the MAPE-K reference model. Moreno et al. [24] introduced the concept of uncertainty reduction to manage uncertainty in self-adaptive systems and show how uncertainty reduction decisions can be integrated into self-adaptation decisions. Our work addresses some challenges put forward by Moreno et al. Besides following a MAPE-K adaptation approach, Parley focuses on allowing an explicit representation and the correct-by-construction synthesis of uncertainty controllers. In particular, employing the estimate of the uncertainty-aware controller, Parley can act on handling uncertainties if and when necessary. We believe the work by Camara et al. in [5] is the closest to ours. In that work, they present a formal reasoning technique based on stochastic multiplayer games to improve decision-making through uncertainty-aware and uncertainty-ignorant decision-making in regions of the state space in which aleatoric uncertainty matters. Parley also benefits from such a separation of uncertainties for the controllers together with a formal technique underpinning our uncertainty reduction service in the formalism of pDTMC. Additionally, Parley resorts to a meta-heuristic approach to find near-optimal adaptation policies. Kreutz et al. [17] recently proposed a new approach to model uncertain knowledge to estimate the best adaptation tactic. We argue that their paper, again, shows the need for adaptive uncertainty resolution to best utilise their modelling notation.

Planning-based approaches: Various planing-based approaches for tackling the decision-making process under uncertainties have also been proposed in the literature [1, 3, 8, 22, 25, 27]. Partially Observable Markov Decision Processes (POMDP) have been extensively used: (i) in the robotic domain to reason for imperfect robot actions and environment observations [19], (ii) in partially-observable domains for online planning in the belief space of long-endurance missions [1], (iii) to deal with partial satisficements in environment changes [25], (iv) the human-robot uncertainty interaction problem [22] or (v) on the decision-making process for SASs while offering awareness of non-functional requirements' priorities at runtime through a vector-valued reward function [27]. Despite the inherent ability of POMDP to robustify systems in the presence of uncertainty, POMDP planning is computationally intractable in the worst case. Moreover, our approach goes one step further through the separation of concern between the adaptive behavior and resolving (epistemic) uncertainty in the URC layer. Esfahani et al. [8] propose POISED, in which a "possibility" distribution is used for tackling the complexity of automatically making adaptation decisions under internal uncertainty. Parley goes one step further by encountering trade-offs, particularly when the probability of satisfying the system's objectives falls under a certain threshold, even without prior knowledge of the probability distribution of the monitored phenomena required by POISED.

3 RUNNING EXAMPLE

Throughout this paper, we use a simplified discrete mobile robot as a running example, navigating within the constraints of a known 10x10 discrete grid map. Maps of similar size have been investigated in literature [11]. The robot can move North, South, East, and West, without leaving the map. The robot's primary mission is to traverse from its initial location to a specified destination without crashing into static obstacles. Such a crash might damage the robot and is hence considered a mission failure. Figure 2 visualises such a map, with the robot starting in the lower left corner, the destination set to the upper right corner, and walls as static obstacles.

To facilitate the robot's movement, we construct a *movement controller* that leverages Dijkstra's shortest path algorithm. Each move incurs a fixed cost (e.g., due to energy consumption). To discourage the robot from venturing too close to obstacles, we introduce penalties for cells near obstacles. To optimise its performance at runtime, we pre-compute moves for every conceivable position on the map, as indicated by the red arrows in Figure 2. At runtime, the movement controller selects moves based on its estimate of the robot's location $(\hat{x}, \hat{y})$.

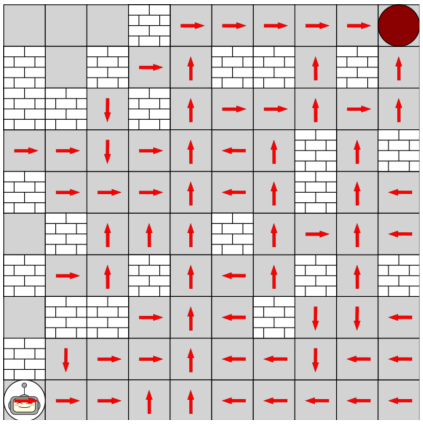


Figure 2: A sample map for our running example. The robot starts in the lower left corner and its mission is to traverse to the destination in the upper right corner without crashing into static obstacles. Red arrows denote move commands.

We consider that each move can have uncertain outcomes, for instance, due to wheel slip [6]. However, the robot perceives all moves as successful, updating its estimate accordingly. Thus, inherent uncertainties can lead to discrepancies between the robot's estimated location and its actual location (x, y) after any move.

A notable challenge arises from the controller's reliance on its estimated location when planning the next move: Eventually, this poses a risk for the robot to crash into an obstacle or not reach the specified destination.

To reduce this uncertainty, a *localisation service* is available that aligns the robot's estimated location with its actual location. We assume that this service has a cost equivalent to five moves. The concrete problem is to determine the frequency c with which the localisation service should be used concerning the estimated location to minimise cost while maximising the mission success rate. A policy for invoking the localisation service can be modelled as a function $f : (\hat{X}, \hat{Y}) \rightarrow C$ where $(\hat{X}, \hat{Y})$ represents the set of possible estimates about the robot's location and C denotes a frequency for invoking the service.

The objectives for each mission are twofold: to successfully reach the destination and to minimise cost. We define the following objectives accordingly:

(O1): **Mission success:** The probability of reaching the destination should be maximised, and
(O2): **Cost:** The mission cost of moving and invoking the localisation service should be minimised.

We evaluate policies based on these objectives. Striking a balance between these objectives aims to optimise the robot's efficiency, ensuring precise navigation with resource conservation in mind.

We use a discrete mobile robot as a running example that navigates through a discrete map. The robot's moves suffer uncertain outcomes, and the location can only be estimated accurately by invoking a localisation service. Essentially, our goal is to determine how the frequency with which the localisation service is invoked should ideally be adapted to minimise mission costs while maximizing the mission success rate.

4 PARLEY

In this section, we present the Parley methodology to synthesise dedicated controllers that reduce uncertainty with formal guarantees. First, we discuss the architecture of a system employing Parley before describing the process of Parley, including assumptions on the underlying system, synthesis of the new controller, computing formal guarantees, and finally its behaviour at runtime. We illustrate these points in the running example of a mobile robot introduced in the previous section. Afterwards, we elaborate on a tool that instantiates the Parley methodology to automatically generate a URC and synthesise policies which the stakeholders can choose from.

4.1 Architecture

To separate the concerns within an adaptive or autonomous system, we propose to emphasise uncertainty reduction by establishing it as a dedicated controller, bringing it on par with controllers that are concerned with adapting the managed system's behaviour.

In Parley, the existing notion of a *change management layer* (cf. [16]) is handled by two separate controllers, as visualised in Fig. 1: First, an *uncertainty-aware controller* (UAC) that employs the existing notion of adapting the managed system. This controller works distinctly with the estimates of variables, clearly incorporating uncertainty. Secondly, a novel *uncertainty reduction controller* (URC) is solely concerned with mitigating epistemic uncertainty to aid the UAC in its decision-making by adapting when and how the UAC monitors the managed system and the environment.

With our two-layer control architecture, we propose a separation of concerns, with the UAC reducing uncertainty in its estimates at a fixed frequency and the URC dedicated to adapting this frequency dynamically to optimise the system's objectives.

Example: In our running example, the movement controller is the UAC, controlling the robot's movement using its estimated location and invoking the localisation service with a fixed period. The URC adapts the frequency with which the localisation service is invoked to reduce uncertainty based on the estimated location.

4.2 Process

In the remainder of this section, we elaborate on the Parley methodology step-by-step from assumptions on the underlying system, to the controller synthesis, until we finally describe its runtime behaviour. The inputs to our automated process are:

- a system model depicting the environment, UAC, managed system and uncertainty reduction services,
- a range for thresholds or frequencies with which these services can be employed,
- a list of variables that the URC relies on for its control, and
- formalised system objectives;

to produce the following output:

- an extended system model with a URC,
- a list of Pareto-optimal policies to trade off when to employ the uncertainty reduction services.

4.2.1 Assumptions on the Uncertainty-Aware Controller. First, we begin by clarifying assumptions about the UAC and the managed system. We assume that the UAC and the managed system are modelled as a discrete-time Markov chain (DTMC) M that encodes the behaviour of the system, *and* its adaptation strategies. M consists of states, a transition matrix P that depicts probabilities of transitioning from one state to another and an initial state. Each state in M is a tuple

$$state = (s, \hat{z}, z),$$

where s corresponds to information fully known to the UAC, and $\hat{z}$ is the UAC's estimate of the information z that the controller does not know. Due to this uncertainty, the UAC makes its decisions solely based on s and $\hat{z}$. Transitions over these states, defined by a transition probability function P, depict the dynamic behaviour of the system. We further assume that the controller's estimate $\hat{z}$ can be optimised using uncertainty reduction services[3]. The controller naively invokes the services, that is, for example with fixed frequencies, or when some variable is below a threshold. We assume that these frequencies or thresholds are encoded in a constant vector, namely c such that c is of dimension n if n uncertainty reduction services are available.

Additionally, we assume that there are formalised objectives for the system. Typically, these refer to functional (e.g., success rate) or non-functional (e.g., costs, performance) properties. As is typical for DTMCs, a reward structure assigning rewards to selected states can be useful for modelling non-functional properties. Probabilistic model checking can be used to calculate the probability with which a property is satisfied by M or to calculate the estimated accumulated reward of M (see later steps).

> We assume a self-adaptive system, with a controller relying on its estimate of variables suffering uncertainty to adapt the managed system. The controller naively invokes services to reduce the uncertainty in its estimate, e.g., using fixed frequencies or thresholds c. We assume that the system, including its controller and environment, is modelled as a DTMC.

Example: For our robotic example, we construct a DTMC with PRISM [18] (cf. Lst. 1). We employ PRISM's notion of modules which synchronise over labelled transitions, i.e., the transition *east*, cf. ll. 8, 17, 28, can be invoked only if all three modules can invoke the transition, invoking it in one single step and updating the rewards accordingly (cf. l. 36). In the model's first module, we depict the ground truth $z = (x, y)$ (cf. ll. 5-14) resembling the robot's actual location, which changes when the robot moves. For the second module, we employ Dijkstra's shortest path algorithm to construct the movement controller, which resembles the UAC, as explained in Sec. 3. This module (cf. ll. 16-20) controls which move the robot should make depending on its estimated location $\hat{z} = (\hat{x}, \hat{y})$, as depicted, e.g., in l. 17. Finally, the controller's knowledge is handled by a dedicated module that includes the estimated location, cf. ll. 24-25. Additionally, a *step* counter is modelled in the Knowledge (cf. l. 26), which is incremented after every move of the robot (cf. l. 32). A localisation service is available (cf. Sec. 3) that aligns the estimated location with the actual location. It is invoked if *step* exceeds a constant limit c (cf. l. 22), which may have been set to, e.g., $c = 2$. *step* is reset (cf. transition *localisation*, l. 31), accordingly. In this sense, *step* is part of the information fully known to the UAC, s. Note, that any movement or invocation of the localisation service incurs a cost (cf. ll. 35-40). We employ the objectives discussed in Sec. 3, accordingly.

```
1  dtmc
2  const int N = 9; //map size
3  const double p = 0.01; // probability of deviation in moves
4
5  module Robot
6    x : [0..N] init 0;
7    y : [0..N] init 0;
8    [east] true ->
9      (1-3*p): (x'=min(x+1, N)) +
10     p: (y'=min(y+1, N)) +
11     p: (y'=max(y-1, 0)) +
12     p: (x'=max(x-1, 0));
13   ...
14 endmodule
15
16 module Adaptation_MAPE_Controller
17   [east]  (x̂=0) & (ŷ=0) -> true;
18   [north] (x̂=1) & (ŷ=0) -> true;
19   ...
20 endmodule
21
22 const int c = 2;
23 module Knowledge
24   x̂ : [0..N] init 0; //estimated position
25   ŷ : [0..N] init 0; //estimated position
26   step : [1..10] init 1;
27   ready : Bool init true;
28   [east]  ready -> (x̂'=min(x̂+1, N)) & (ready'=false);
29   [north] ready -> (ŷ'=min(ŷ+1, N)) & (ready'=false);
30   ...
31   [localisation] step>=c & !ready -> (x̂'=x) & (ŷ'=y) & (step'=1) &
         (ready'=true);
32   [skip] step<c & !ready -> (step'=step+1) & (ready'=true);
33 endmodule
34
35 rewards "cost"
36   [east] true : 1;
37   [north] true : 1;
38   ...
39   [localisation] true : 5;
40 endrewards
```

Listing 1: Excerpt of PRISM code for the robot's movement.

4.2.2 Synthesising the URC. We propose to synthesise a dedicated controller, the URC, to control when the UAC employs its uncertainty reduction services. To this end, the URC adapts the vector that depicts the frequencies of updates or thresholds c. The URC performs these adaptations based on variables of the UAC: s and $\hat{z}$. We propose that the designers select which of these variables the URC can use to decide how it should adapt the UAC[4]. Note, that adaptation decisions cannot be made based on the ground truth as the ground truth is unknown to the controllers. We depict possible adaptation decisions based on the selected set of variables as a function $decision(s, \hat{z})$ for any value in the selected subset of s and $\hat{z}$, accordingly, that computes a vector of dimension n if n uncertainty reduction services are available. To synthesise the URC, we extend the DTMC such that c can be set dynamically during any run, corresponding to the desired frequencies or thresholds.

[3] Often only one such service will be available and the optimised estimate will be z itself.

[4] Usually, a subset of these variables will be sufficient to perform adequate adaptations.

An augmented state is now a tuple

$$state_{aug} = (s, \hat{z}, z, c).$$

where c depicts the frequencies of updates or thresholds. We augment the transition matrix P accordingly, such that the transition probability for any augmented state $(s, \hat{z}, z, *)$[5] is set to the corresponding transition probability in P, and c is set according to the *decision* function, i.e.,

$$P'((s, \hat{z}, z, c), (s', \hat{z}', z', c')) :=$$

$$\begin{cases} P((s, \hat{z}, z), (s', \hat{z}', z')) & decision(s', \hat{z}') = c' \\ 0 & \text{otherwise} \end{cases}$$

Therefore, any of the existing transitions, including the UAC's adaptations of the managed system, remain unchanged. P' reflects solely the desired selection of c in every transition, according to the values selected by the *decision* function. Since the *decision* function has not been selected yet, the concluding model M' becomes a parametric DTMC (pDTMC). Concrete values for these parameters will be determined by probabilistic model checking (see next step). One concrete instance of this function, assigning concrete values to c, is called a *policy*.

For DTMC models in the PRISM modelling language, we provide a tool that automatically performs this augmentation to pDTMC models, using the set of variables that a decision should be based on, as well as transitions that are supposed to happen before and after the adaptations. The latter input ensures that the URC only adapts the UAC when it is in a quiescent state and that the augmentation does not introduce any non-determinism. Our tool then automatically adds parameters depicting the *decision* function, as well as a new module, called *Uncertainty_Reduction_Controller* to the PRISM file making the model a pDTMC. Listing 2 showcases such an extension: the variable *turn* is used to depict when the UAC is quiescent. For practical reasons, we enforce an upper bound on c, which can be adapted to any value in the defined range (cf. l. 5).

We synthesise a URC that is responsible for adapting the frequencies or thresholds c with which uncertainty reduction services are invoked. The adaptation decisions are modelled as parameters, resulting in a parametric DTMC.

Example: In our example, we automatically add a module *Uncertainty_Reduction_Controller* (cf. ll. 4-13 in Lst. 2) to the model and move the variable c to this module (cf. l. 5 in Lst. 2)[6]. Hence, the URC can adapt c. To depict quiescent states in which an adaptation is permitted, we use *turn* (cf. l. 6 in Lst. 2) that sequentially interrupts the system and UAC to allow for adaptations performed by the URC, i.e., when $turn = 2$ (cf. l. 9). In our example, each simulation alternates between movements, adaptations, and finally invoking or skipping the localisation service. We choose that potential adaptations of the URC depend only on the estimated location $\hat{x}$ and $\hat{y}$. Thus, the decision function depicts the URC's adaptations as parameters for any possible value of $\hat{x}$ and $\hat{y}$, i.e., $decision\_\hat{x}\_\hat{y}$ depicts $decision(\hat{z})$ with $\hat{z} = (\hat{x}, \hat{y})$ (cf. l 2). We leave the parameters without concrete values since no policy has been defined yet.

[5]With $*$ denoting a wildcard, depicting that the probability does not depend on c.
[6]L. 22 in Lst. 1 is deleted, accordingly.

```
1  ...
2  const int decision_0_0; //decision to invoke service at (0, 0)
3  ...
4  module Uncertainty_Reduction_Controller
5    c : [1..10] init decision_0_0;
6    turn : [1..3] init 1;
7    [east] turn=1 -> (turn'=2);
8    ...
9    [] turn=2 & x̂=0 & ŷ=0 -> (c'=decision_0_0) & (turn'=3);
10   ...
11   [localisation] turn=3 -> (turn'=1);
12   [skip] turn=3 -> (turn'=1);
13 endmodule
```

Listing 2: Excerpt of the PRISM code depicting the URC adapting c based on the estimated location, automatically synthesised based on Listing 1.

4.2.3 Generating Policies with Guarantees. To find suitable policies that can satisfy the specified objectives for the pDTMC constructed in the previous step, we employ probabilistic model checking. With probabilistic model checking, we can evaluate a particular policy. If the number of possible policies is small enough, an extensive evaluation of every possible policy can be performed. In case the potential number of policies is very large, however, we propose to resort to a meta-heuristic approach that finds near-optimal policies. While this does not provide guarantees for optimality, invoking a probabilistic model checker such as PRISM [18] guarantees the near-optimal trade-offs between objectives that have been identified by the meta-heuristic search.

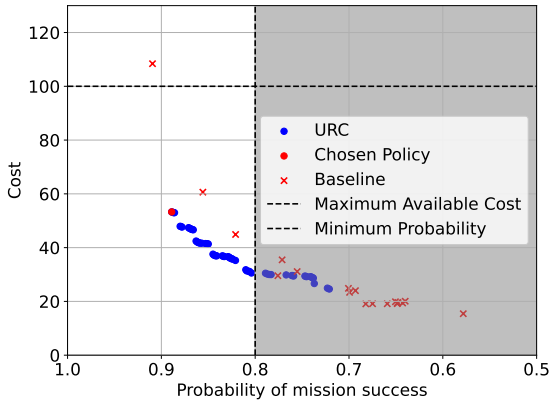
Policies define values for the parameters and reflect the URC's behaviour. These policies can be found by employing probabilistic model checking of the pDTMC. If the objective space is too large, meta-heuristic search can be applied.

Example: In our running example, each policy consists of 100 parameters, one for each possible (estimated) location on the map (100 cells in the 10x10 map). We choose to limit the maximum interval between two invocations of the localisation service to ten steps due to the amount of obstacles that are present on the maps and the resulting high likelihood of a crash. Since each parameter can denote any interval between 1 and 10, the number of possible policies is 10^{100}. Due to the large search space, we resort to EvoChecker [10], a meta-heuristic approach to finding policies. Figure 3a visualises the Pareto-front of policies that EvoChecker provides for the given map (cf. Fig. 2) and guaranteed trade-offs of the objectives.

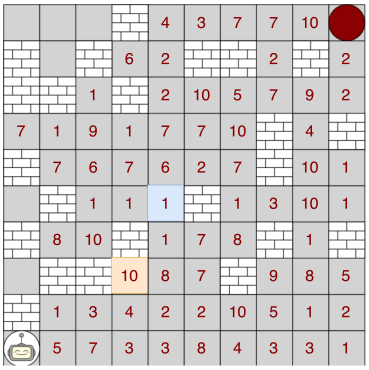
4.2.4 Selecting a Policy for the URC. The policies identified in the previous step form a Pareto-front concerning the specified objectives. At runtime, the stakeholders can trade off the objectives and select one of the policies of the Pareto-front accordingly. This trade-off is essential as it enables the stakeholders to prioritise the objectives, i.e., the requirements. Potential techniques to automatically select a policy, for instance with the knee method, can be used. The selected policy is deployed in the URC.

Stakeholders select a policy from the Pareto-front by trading off objectives and prioritising requirements, accordingly.

Example: In our robotic example, we encounter that the robot only has limited energy available during its mission. Therefore, we only consider policies that incur a cost of less than 100. Additionally,



(a) Pareto-front of policies generated by EvoChecker. The stakeholders determine a maximum cost of 100 and select one policy, accordingly.



(b) Extension of the map shown previously. The selected policy is visualised by numbers showcasing the intervals between localisations.

Figure 3: Policy selection for the robotic example.

the stakeholders decide that the probability of a successful mission should exceed 80%. Fortunately, the Pareto-front contains policies satisfying these constraints. Otherwise, the stakeholders would have to compromise and prioritise one of the requirements, for example, if the map contains so many obstacles on the robot's main path to its destination that the chance of a crash is greater than usual. In our example, however, we choose the policy depicted by a red dot in Figure 3a as it guarantees the highest mission success probability while staying below a cost of 100. The parameters provided by this policy are displayed in Figure 3b. Every identified policy can be found in our publicly accessible online repository[7].

4.2.5 Enacting and Executing the Selected Policy. At runtime, the URC monitors the variables chosen for its decision-making. Using the selected policy, it adapts the frequencies or thresholds c in the UAC.

The selected policy is invoked at runtime and the URC adapts when the uncertainty reduction services are invoked.

Example: In the robotic example, the URC updates the intervals between invocations of the localisation service by the UAC at runtime. We make the following observation: The URC adapts c such that shorter intervals are used when the robot's planned path has an obstacle nearby, e.g., at location (4, 4) (shaded blue cell in Fig. 3b). However, in locations that the robot only steps into due to deviations in its movement, such as (3, 2) (shaded orange cell in Fig. 3b), the interval is set to the maximum. We assume that this is because the robot can only estimate that it is in this location when it has performed a localisation anyway. Thus, the URC helps to reduce the use of resources while maintaining a high probability of a successful mission (cf. Fig. 3a).

4.3 Automated Tool

We instantiate the PARLEY methodology and provide an automated tool[8] that performs the steps described in the previous subsections. Our tool uses the following inputs:

- a PRISM file containing a DTMC depicting the managed system, environment (ground truth) and URC with its estimates,
- transition labels occurring *before* adaptations from the URC,
- transition labels occurring *after* adaptations from the URC, and
- a list of variables which are used for the URC's decision.

to automatically add modules to the PRISM file depicting the URC, as shown in Listing 2. Afterwards, our tool automatically calls EvoChecker and uses objectives defined in PCTL to provide a list of Pareto-optimal policies which the stakeholders can choose from.

5 EVALUATION

To evaluate PARLEY, we formulate the following research questions that we investigate on the robotic use case outlined previously.

RQ1: **Effectiveness:** How effective is PARLEY's adaptive uncertainty reduction in terms of achieving the system's objectives (success rate and costs) compared to a baseline that resolves uncertainty periodically?

RQ2: **Diversity:** How diverse are the policies provided by PARLEY to enable trading off multiple objectives compared to the baseline?

RQ3: **Scalability:** How scalable is PARLEY when increasing the model size (size of the map)?

To evaluate the practicality of PARLEY, we formulate an additional research question:

RQ4: **Practicality:** Does PARLEY work in a realistic setting of a robotic use case, and is PARLEY applicable to other types of systems?

[7] https://github.com/carwehlm/PARLEY/tree/main/plots/fronts

[8] https://github.com/carwehlm/PARLEY

Experimental Setup for RQ1 and RQ2: To evaluate the effectiveness and diversity of Parley on the robotic use case (Sec. 3) according to the PRISM model discussed in Sec. 4, we randomly generate 90 maps of size 10x10. The robot should traverse from the bottom-left to the upper-right corner of the map. Obstacles are placed randomly on the map[9]. Based on the 90 individual maps and their movement controllers, we synthesise 90 PRISM models, as discussed in Sec. 4. In this context, uncertainty occurs due to non-deterministic outcomes of the robot's movements and it can be reduced by using a localisation service to obtain the actual location of the robot (cf. Sec. 3). We use the objectives defined in Sec. 3 accordingly.

We compare Parley to a *baseline* on each of the 90 PRISM models denoting 90 *scenarios* of the robotic use case. The baseline generates 10 policies that invoke the localisation service with fixed frequencies (i.e., periodically). The first policy invokes the localisation service after each movement of the robot, the second policy after every other movement, etc.[10] The baseline policies form a Pareto-front concerning the objectives, as depicted by the red crosses in Fig. 3a. In contrast, Parley generates policies that adapt the frequency with which the localisation service is invoked depending on the robot's estimated location. Policies found by Parley also form a Pareto-front, as depicted by the circles in Fig. 3a.

To quantify the effectiveness (RQ1), we compute the *Hypervolume* [20] that measures the size of the objective space covered by the Pareto-front obtained by each approach. Hypervolume is considered a comprehensive quality indicator for the convergence, diversity, and cardinality of a solution set that is applicable if the number of objectives is rather low, which is true in our case. It can be used to compare two solution sets while a higher Hypervolume indicates a "better" set in terms of Pareto dominance [20]. To quantify the diversity (RQ2), we compute the *Spread* [20] of a Pareto-front obtained by each approach. Spread is a quality indicator dedicated to the diversity of solutions on a Pareto front. It is applicable to bi-objective problems and measures the distribution and uniformity of the solutions in a front. A smaller value is preferred, indicating a better distribution [20]. For the a-posterior analysis of the Pareto-fronts obtained by Parley and the baseline, we compute the Hypervolume and Spread concerning practically relevant solutions, which are constrained by individual requirements on the two objectives. Particularly, we denote with $min\_success \in \{0.6, 0.7, 0.8\}$ the minimal success rate (Objective O1 from Sec. 3) and with $max\_costs \in \{100, 80, 60\}$ maximal costs that are required (Objective O2 from Sec. 3). We consider all combinations of these two requirements, resulting in nine *settings* to cover a wide range of practically relevant solutions. We depict one such setting with dotted lines in Fig. 3a and only investigate policies that are within the area of acceptable policies (non-shaded area).

To address the stochastic nature of the metaheuristic search (EvoChecker[11]) in Parley, we run Parley ten times on each map. In contrast, the baseline is deterministic so that one run is sufficient. We further quantify the gain of Parley concerning Hypervolume and Spread, and test for statistical significance of the gain using Mann-Whitney U and a 95% confidence level ($p<0.05$) (cf. [2]).

Experimental Setup for RQ3: To investigate Parley's scalability, we employ the same setup of a robotic use case described previously but scale the map size from 5 to 20 in steps of 5 to analyse how large the DTMC's state space is, how quickly PRISM can verify the objectives to evaluate a policy[12], and finally how large the objective space is for Parley.

Experimental Setup for RQ4: We investigate RQ4 in two dimensions: We deploy Parley to *ROS Gazebo*[13] to demonstrate its feasibility in a realistic setting. We further apply Parley to a self-protecting web application used in the literature [24] to demonstrate its applicability to a different type of system.

5.1 RQ1: Effectiveness

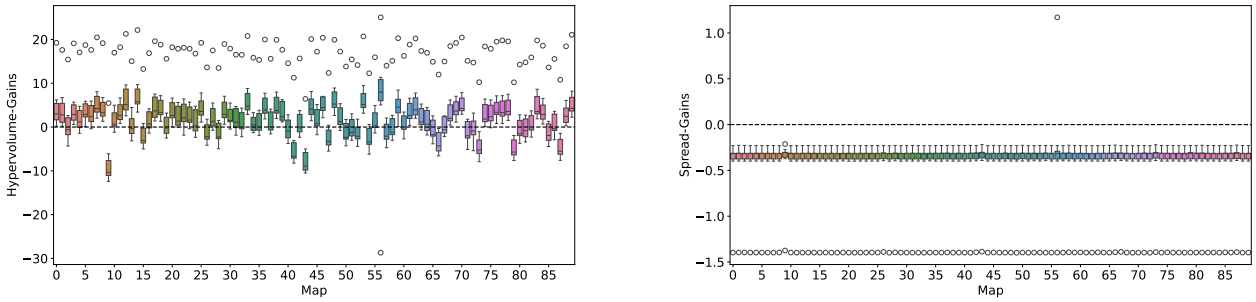
For the first research question, we investigate if the policies provided by Parley are better, that is, closer to the optimum (minimal cost and maximum probability of a mission success), than those provided by the baseline. Table 1 reports the results in terms of Hypervolume gains achieved by Parley over the baseline in the upper entries of each row. For instance, for a minimal success rate of 70% and maximal cost of 80, Parley significantly outperforms the baseline on 48 out of 90 maps, it is significantly outperformed by the baseline in 11 maps, and the results for the remaining 31 maps are similar (i.e., no statistically significant difference). Most importantly, we can observe that Parley improves its performance over the baseline when we strengthen the requirements, that is, we increase the required minimal success rate (*min_success*) and reduce the maximal cost (*max_cost*). For strong requirements of $min\_success \geq 80\%$ and $max\_cost \leq 60$, Parley significantly outperforms the baseline in 60 out of 90 maps. In contrast, for weak requirements of $min\_success \geq 60\%$ and $max\_cost \leq 60$, the baseline performs better than Parley on more maps. This shows that for cases with weak requirements, a naive approach such as the baseline can be used, while cases with stronger requirements demand a smart approach such as Parley. Thus, on average across all nine requirement settings (each with 90 maps), Parley provides statistically significant improvements in 405 out of 810 cases. Figure 4a visualises the results for each map for one setting ($min\_success \geq 70\%$ and $max\_cost \leq 80$)[14].

> We conclude that Parley significantly helps to achieve the system's objectives by determining when to reduce uncertainty in 50% (405/810) of the cases compared to the baseline. Most importantly, it showed increased performance improvements over the baseline when strengthening the requirements. In such settings, Parley would be preferred over the baseline.

5.2 RQ2: Diversity

For the second research question, we investigate if the policies provided by Parley provide more diverse trade-offs than those

[9] For each cell, we generate a random number with a standard normal distribution. If the number is outside of $[-\sigma, \sigma]$, an obstacle is placed in the cell. For every map we used, we checked that there exists a path from the robot's start to the destination.
[10] As discussed in the previous section, we set the maximum interval between two invocations of the service at 10 steps.
[11] We use EvoChecker out of the box and set a population size of 100 across 40 generations.
[12] We use PRISM in version 4.7 on an M1 MacBook Pro with 16GB RAM.
[13] Gazebo is a simulator for systems based on the Robotic Operating System (ROS).
[14] We provide plots for other settings in our replication package https://github.com/carwehlm/PARLEY/tree/main/plots/box-plots .



(a) Gain in Hypervolume of Parley, higher is better. **(b) Gain in Spread of Parley, lower is better.**

Figure 4: Gain in (a) Hypervolume and (b) Spread of Parley compared to baseline across different maps in requirement setting $min\_success \geq 70\%$ **and** $max\_cost \leq 80$**.**

Table 1: Results for gains in Hypervolume (HV) and Spread (SP) of Parley compared to baseline in all settings, denoted by three values: the number of maps in which Parley is significantly better / significantly worse / insignificant compared to the baseline.

min_success		max_cost		
		60	80	100
60%	HV	29/34/27	36/27/27	37/22/31
	SP	90/00/00	89/01/00	89/01/00
70%	HV	46/11/33	48/11/31	45/11/34
	SP	90/00/00	90/00/00	90/00/00
80%	HV	60/12/18	54/12/24	50/11/29
	SP	86/00/04	86/02/02	86/02/02

provided by the baseline. If so, more different trade-offs along the Pareto front rather than similar trade-offs are available and can be selected by stakeholders for decision-making, which results in more diverse options for self-adaptation. To this end, we compute the Spread (a lower value indicates a higher diversity). Table 1 reports these results in the lower entries of each row for the nine requirement settings. For instance, for a minimal success rate of 70% and maximal cost of 80, Parley significantly outperforms the baseline on all 90 maps. For all settings, Parley achieves significantly better results for at least 86 out of 90 maps, and on average, across all nine settings, Parley provides statistically significant improvements in 796 out of 810 cases. Figure 4b additionally visualises the gain in Spread (lower is better) achieved by Parley over the baseline for one setting ($min\_success \geq 70\%$ and $max\_cost \leq 80$).

We conclude that the policies provided by Parley are significantly more diverse than the baseline policies in 98.3% (796/810) of the cases, which offers a more diverse set of trade-offs, from which stakeholders can select for self-adaptation.

5.3 RQ3: Scalability

In the third research question, we are concerned with the scalability of our approach. Multiple facets need to be considered here:

- **State space:** The state space of the pDTMC determines how long the evaluation of a policy with a model checker such as PRISM takes,
- **Search space:** The domains of variables that are available to make a decision, as well as all possible decisions, determine the search space of possible solutions.

We investigate different map sizes of the running example. Tab. 2 shows the results. We can see an exponential growth in the pDTMC state space, the time to perform model checking grows accordingly. In combination with the exponential growth of the search space for possible solutions, we conclude that our approach might not scale well to larger problems. For a larger map (20x20) and 50 generations with a population size of 50, the model checker would be invoked 2,500 times per objective (property), resulting in a total verification time of $\approx$ 83 minutes (2,500 $\times$ 1s $\times$ 2 properties). In contrast, model checking every policy with PRISM would take $\approx 10^{518}$ minutes ($20^{400} \times$ 1s $\times$ 2 properties). Thus, Parley improves the scalability, which, however, can still be an issue for applying Parley at runtime in highly dynamic systems.

Scalability remains an issue for Parley but might be tackled by more abstract models, more computational resources, or a reduced search space for policies.

5.4 RQ 4: Practicality

To investigate the final research question, we first apply Parley in a realistic robotic setting. Afterwards, we investigate how Parley can be applied to the self-protecting web application.

Table 2: Scalability of Parley. With larger maps, the pDTMC's state space (#S) increases as well as the time of model checking in PRISM for the two properties O1 and O2, and the search space of possible policies.

Map size	#S	PRISM (s)	Search space
5x5	2,413	0.01, 0.032	$5^{25} \approx 10^{17}$
10x10	2,977	0.092, 0.067	10^{100}
15x15	6,593	0.313, 0.297	$15^{225} \approx 10^{264}$
20x20	11,276	0.864, 0.897	$20^{400} \approx 10^{520}$

Continuous Robot: Expanding from the problem investigated so far we consider a more realistic setup. Specifically, a Turtlebot3 Waffle[15] must travel to a destination while not crashing into obstacles or leaving the area. The system now contains a continuous robot concerning its position. The robot now also has a heading angle, θ, discretised into four directions: North, East, South, and West.

For modelling purposes, the environment is discretised similarly to before. The environment is $21mx21m$ discretised into $3mx3m$ cells, resulting in a $7x7$ environment. The heading angle also suffers uncertainty, hence the controller works with an estimate that can become inaccurate over time. Therefore, the controller's estimates are as follows:

$$\hat{z} = (\hat{x}, \hat{y}, \hat{\theta})$$

The movement controller determines the shortest path as described earlier and executes the series of commands based on its estimates. However, the left wheel is slightly faulty, operating at 99% power. This means the robot will drift over time. For this problem, the robot has no sensors but can use a localisation service similar to the running example. The system has the same objectives as the running example. We set the maximum frequency between two localisations to five given the robot's high drift and apply Parley to generate a Pareto-front of policies, visualised in Fig. 5a. Parley provides 30 Pareto-optimal policies compared to just five policies found in the baseline.

We conducted simulations in Gazebo using ROS packages. To acquire the probability transitions the robot was first initialised in a cell, and chose random actions (North, East, South, West) until the robot crashed or left the environment. These traces were recorded to attain the state transitions and the corresponding probabilities. The code for running the experiments can be found online, along with a video capturing the simulation[16].

Web Application: Moreno et al. [24] presented a web application containing a web server (A) and a database (B). They motivate that an attacker might infiltrate A and that the system uses an intrusion detection system (IDS) to receive alerts about such cyber-attacks. If the attacker is successful, A is compromised, and the attacker might go on to infiltrate B as well. In case B, too, becomes compromised, a high cost is assumed due to the sensitivity of the data in B. A controller has the options to restore A to a non-compromised version (*restoreA*), restore both servers to non-compromised versions (*restoreAB*), not intervene at all (*NOP*), or to reduce the uncertainty in the IDS alert and determine accurately if A is compromised *scanA*. Note, that while all actions except *NOP* reduce uncertainty in some way, we only consider *scanA* as an uncertainty reduction service as the other actions have additional implications on the system other than to solely reduce uncertainty, i.e., restoring a server.

We modelled such a system with an extension to include dynamic costs of restoring either server A or both servers to depict that there might be times (e.g. during the night) when the server's downtime is not as expensive as at other times. Additionally, we model variability in the IDS to depict that the quality of its predictions may be dynamic, i.e., when the IDS's rule set has not been updated recently, its quality deteriorates.

[15]https://www.turtlebot.com/turtlebot3/

[16]https://github.com/carwehlm/PARLEY/blob/main/turtlebot.mp4

Table 3: Key characteristics of the systems and pDTMC models used for the evaluation.

	Discrete Robot	Continuous Robot	Web Application
Application Domain	mobile robot navigation		server infrastructure management
System Type	CPS	CPS	multi-server system
pDTMC			
# states	2,977	512	5,630
# transitions	4,779	1,070	11,540
# parameters	10^{100}	10^{7*7*4}	132

Similar to the original model, we use the IDS's prediction of an attack to determine the controller's action. However, we employ Parley to decide **when** *scanA* should be invoked to reduce uncertainty, depending on the **cost of restoring the servers** and the **IDS's confidence**. In contrast to previous examples in this paper, we now determine thresholds instead of frequencies at which the uncertainty reduction service is invoked. Fig. 5b displays the Pareto-front of policies generated by Parley and the baseline solution, which does not consider the dynamic cost or the IDS's confidence w.r.t. the cost of an infected database (x-axis) and cost of controlled actions (y-axis). Parley provides 139 Pareto-optimal policies compared to just a single one provided by the baseline.

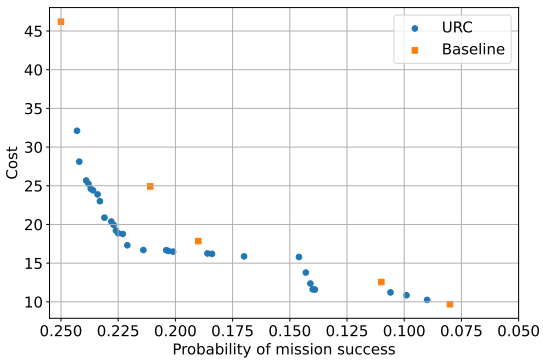
Summary: Table 3 displays the main characteristics of all systems we investigated. While the robots are cyber-physical systems (CPS), the web application provides initial evidence that Parley is further applicable to another application domain. Thereby, with the robots we considered systems with larger search spaces (# parameters), and with the web application we considered a system with a larger state space (# states and # transitions), which shows the practicality of Parley in different settings.

We have successfully applied Parley to two further systems: a realistic use case of a real continuous robot (Turtlebot3), and a web application from literature. Parley was able to find more policies than the baseline in both applications: 6 times more for the continuous robot and 139 more for the server system. This provides initial evidence about the practicality of Parley in different domains and complexity of systems.

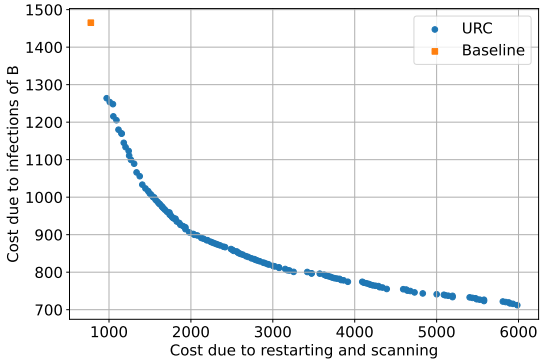
5.5 Threats to validity

Threats to the validity of our study are as follows:

Internal: The analysed maps for RQ1 and RQ2 pose a threat to internal validity. We generated 90 different maps randomly to mitigate this threat. Additionally, we conducted ten runs of EvoChecker for each of the maps to account for its meta-heuristic nature. For RQ3, the investigated maps might not be representative of maps of the selected sizes. The same holds for RQ4, where we merely demonstrated the applicability to other applications, but did not investigate different problem instances in these domains. We refer to the extensive literature on probabilistic model checkers, e.g. [13, 18] for more information on their scalability. Currently, Parley uses



(a) Continuous Robot.



(b) Web Application.

Figure 5: Pareto-fronts for different applications.

PRISM, which limits its scalability. Using EvoChecker [10], we started addressing this concern but still limited the map size. Additionally, we modelled the uncertainty reduction services as minimalistic mocks that rely on the ground truth. This will, usually, not be possible in reality. Further, we might have made mistakes in modelling any of the investigated systems. We reviewed the models internally and make them accessible publicly online. Finally, we used EvoChecker out of the box and did not experiment with tuning its hyperparameters.

External: We selected three different systems from two different domains (cf. Tab. 3) to mitigate the threat of generalisability. We further plan to investigate additional systems in the future.

Conclusion: We used the Mann-Whitney U test for significant differences between Parley and the baseline with a 95% confidence level and relied on the extensive guidelines in [2] to conduct our experiments to mitigate this threat. The selection of the requirement settings for RQ1 and RQ2 is, however, a threat to the conclusion validity. We mitigated this threat by considering nine different settings covering all combinations of weaker and stronger requirements for minimal success rate and maximal cost.

6 CONCLUSION AND FUTURE WORK

In this work, we motivated the need for a controller dedicated to reducing epistemic uncertainty. We introduced Parley, an end-to-end methodology and architecture that relies on a pDTMC system model and synthesises an uncertainty reduction controller (URC), clearly separating the concerns between adapting the managed system's functionality and reducing uncertainty. For models in the PRISM language, we created a tool to automate the synthesis of a URC. Parley further relies on probabilistic model checking to provide formal guarantees for the synthesised controller. Given the inherent limited scalability of model checking, we use EvoChecker, a meta-heuristic approach based on PRISM that trades off multiple objectives through Pareto-optimal solutions.

Parley was evaluated on 90 instances of the robotic example with nine different requirement settings, and compared to a baseline concerning effectiveness and diversity. The results show that Parley significantly outperforms the baseline in 50% of all 810 cases in terms of effectiveness while showing improved effectiveness when strengthening the requirements. For the diversity, Parley presented significant improvements compared to the baseline in 98.3% of all 810 cases. Thus, Parley enables a richer and more diverse set of trade-offs, from which the stakeholders can choose for self-adaptation. Finally, we demonstrated Parley's practicality by applying it to three different systems from two domains, including a self-protecting web application from the literature.

In future work, we aim to investigate if a tuning of EvoChecker's hyperparameters can guide the search toward even better results and if different abstractions (models) improve scalability. While we demonstrated Parley's practicality by applying it to three different systems, in future work we aim to apply Parley to more systems of other domains, such as TAS [30]. To this end, we plan to model the controller's estimations with probabilistic distributions. Additionally, we plan to investigate how changes in the model at runtime can be handled by refining selected policies with EvoChecker, e.g. when assumptions about the environment have changed. Our future plans also include applying Parley to other sources of uncertainty, such as those involving system goals and human interactions. A special emphasis will be put on including aleatoric uncertainty in the uncertainty reduction services, i.e., being able to express imperfect sensors and services.

Data availability: We make all models, data, and code publicly available in our reproduction package located at: https://github.com/carwehlm/PARLEY.

Acknowledgements: This work received funding from the Assuring Autonomy International Programme (AAIP). This study was also financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001 and the Alexander von Humboldt Foundation (AvH).

REFERENCES

[1] Ali-akbar Agha-mohammadi, N. Kemal Ure, Jonathan P. How, and John Vian. 2014. Health aware stochastic planning for persistent package delivery missions using quadrotors. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 3389–3396. https://doi.org/10.1109/IROS.2014.6943034

[2] Andrea Arcuri and Lionel Briand. 2014. A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification and Reliability* 24, 3 (2014), 219–250. https://doi.org/10.1002/stvr.1486 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/stvr.1486

[3] 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 (2011), 387–409. https://doi.org/10.1109/TSE.2010.92

[4] Radu Calinescu, Raffaela Mirandola, Diego Perez-Palacin, and Danny Weyns. 2020. Understanding Uncertainty in Self-adaptive Systems. In *IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2020, Washington, DC, USA, August 17-21, 2020*. IEEE, 242–251. https://doi.org/10.1109/ACSOS49614.2020.00047

[5] Javier Cámara, Wenxin Peng, David Garlan, and Bradley Schmerl. 2018. Reasoning about sensing uncertainty and its reduction in decision-making for self-adaptation. *Science of Computer Programming* 167 (2018), 51–69. https://doi.org/10.1016/j.scico.2018.07.002

[6] Nicolas D'Ippolito, Víctor Braberman, Jeff Kramer, Jeff Magee, Daniel Sykes, and Sebastian Uchitel. 2014. Hope for the best, prepare for the worst: multi-tier control for adaptive systems. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 688–699.

[7] Thomas Vogel (Ed.). 2023. Software Engineering for Self-Adaptive Systems exemplars repository. http://self-adaptive.org/exemplars/

[8] Naeem Esfahani, Ehsan Kouroshfar, and Sam Malek. 2011. Taming uncertainty in self-adaptive software. In *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13th European Software Engineering Conference (ESEC-13), Szeged, Hungary, September 5-9, 2011*, Tibor Gyimóthy and Andreas Zeller (Eds.). ACM, 234–244. https://doi.org/10.1145/2025113.2025147

[9] Simos Gerasimou, Radu Calinescu, Stepan Shevtsov, and Danny Weyns. 2017. UNDERSEA: an exemplar for engineering self-adaptive unmanned underwater vehicles. In *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 83–89.

[10] Simos Gerasimou, Radu Calinescu, and Giordano Tamburrelli. 2018. Synthesis of probabilistic models for quality-of-service software engineering. *Automated Software Engineering* 25 (2018), 785–831.

[11] Ruben Giaquinta, Ruth Hoffmann, Murray Ireland, Alice Miller, and Gethin Norman. 2018. Strategy Synthesis for Autonomous Agents Using PRISM. In *NASA Formal Methods*, Aaron Dutle, César Muñoz, and Anthony Narkawicz (Eds.). Springer International Publishing, Cham, 220–236.

[12] Holger Giese, Nelly Bencomo, Liliana Pasquale, Andres J. Ramirez, Paola Inverardi, Sebastian Wätzoldt, and Siobhán Clarke. 2011. Living with Uncertainty in the Age of Runtime Models. In *Models@run.time - Foundations, Applications, and Roadmaps [Dagstuhl Seminar 11481, November 27 - December 2, 2011] (Lecture Notes in Computer Science, Vol. 8378)*. Springer, 47–100. https://doi.org/10.1007/978-3-319-08915-7_3

[13] Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. 2022. The probabilistic model checker Storm. *International Journal on Software Tools for Technology Transfer* 24 (2022), 589–610. Issue 4.

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

[15] Tsutomu Kobayashi, Rick Salay, Ichiro Hasuo, Krzysztof Czarnecki, Fuyuki Ishikawa, and Shin-ya Katsumata. 2021. Robustifying Controller Specifications of Cyber-Physical Systems Against Perceptual Uncertainty. In *NASA Formal Methods*, Aaron Dutle, Mariano M. Moscato, Laura Titolo, César A. Muñoz, and Ivan Perez (Eds.). Springer International Publishing, Cham, 198–213.

[16] Jeff Kramer and Jeff Magee. 2007. Self-Managed Systems: An Architectural Challenge. In *Future of Software Engineering (FOSE '07)*. IEEE, 259–268. https://doi.org/10.1109/FOSE.2007.19

[17] Andreas Kreutz, Gereon Weiss, and Mario Trapp. 2022. Towards Uncertainty Reduction Tactics for Behavior Adaptation. In *European Conference on Software Architecture*. Springer, 199–214.

[18] Marta Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings 23*. Springer, 585–591.

[19] Mikko Lauri, David Hsu, and Joni Pajarinen. 2023. Partially Observable Markov Decision Processes in Robotics: A Survey. *IEEE Transactions on Robotics* 39, 1 (2023), 21–40. https://doi.org/10.1109/TRO.2022.3200138

[20] Miqing Li, Tao Chen, and Xin Yao. 2022. How to Evaluate Solutions in Pareto-Based Search-Based Software Engineering: A Critical Review and Methodological Guidance. *IEEE Transactions on Software Engineering* 48, 5 (2022), 1771–1799. https://doi.org/10.1109/TSE.2020.3036108

[21] Sara Mahdavi-Hezavehi, Danny Weyns, Paris Avgeriou, Radu Calinescu, Raffaela Mirandola, and Diego Perez-Palacin. 2020. Uncertainty in Self-adaptive Systems: A Research Community Perspective. *ACM Trans. Auton. Adapt. Syst.* 15, 4 (2020), 10:1–10:36. https://doi.org/10.1145/3487921

[22] Gonçalo S Martins, Hend Al Tair, Luís Santos, and Jorge Dias. 2019. αPOMDP: POMDP-based user-adaptive decision-making for social robots. *Pattern Recognition Letters* 118 (2019), 94–103.

[23] Rhiannon Michelmore, Matthew Wicker, Luca Laurenti, Luca Cardelli, Yarin Gal, and Marta Kwiatkowska. 2020. Uncertainty quantification with statistical guarantees in end-to-end autonomous driving control. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 7344–7350.

[24] Gabriel A. Moreno, Javier Cámara, David Garlan, and Mark Klein. 2018. Uncertainty Reduction in Self-Adaptive Systems. In *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems* (Gothenburg, Sweden) *(SEAMS '18)*. Association for Computing Machinery, New York, NY, USA, 51–57. https://doi.org/10.1145/3194133.3194144

[25] Luis H Garcia Paucar and Nelly Bencomo. 2018. Re-storm: Mapping the decision-making problem and non-functional requirements trade-off to partially observable markov decision processes. In *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 19–25.

[26] Andres J. Ramirez, Adam C. Jensen, and Betty H. C. Cheng. 2012. A taxonomy of uncertainty for dynamically adaptive systems. In *7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2012, Zurich, Switzerland, June 4-5, 2012*, Hausi A. Müller and Luciano Baresi (Eds.). IEEE Computer Society, 99–108. https://doi.org/10.1109/SEAMS.2012.6224396

[27] Huma Samin, Nelly Bencomo, and Peter Sawyer. 2022. Decision-Making under Uncertainty: Be Aware of Your Priorities. *Softw. Syst. Model.* 21, 6 (dec 2022), 2213–2242. https://doi.org/10.1007/s10270-021-00956-0

[28] Stepan Shevtsov, Danny Weyns, and Martina Maggio. 2019. SimCA*: A Control-Theoretic Approach to Handle Uncertainty in Self-Adaptive Systems with Guarantees. *ACM Trans. Auton. Adapt. Syst.* 13, 4, Article 17 (jul 2019), 34 pages. https://doi.org/10.1145/3328730

[29] Gabriela Félix Solano, Ricardo Diniz Caldas, Genaína Nunes Rodrigues, Thomas Vogel, and Patrizio Pelliccione. 2019. Taming uncertainty in the assurance process of self-adaptive systems: a goal-oriented approach. In *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 89–99.

[30] Danny Weyns and Radu Calinescu. 2015. Tele assistance: A self-adaptive service-based system exemplar. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 88–92.

[31] Danny Weyns, Radu Calinescu, Raffaela Mirandola, Kenji Tei, Maribel Acosta, Amel Bennaceur, Nicolas Boltz, Tomás Bures, Javier Cámara, Ada Diaconescu, Gregor Engels, Simos Gerasimou, Ilias Gerostathopoulos, Sinem Getir Yaman, Vincenzo Grassi, Sebastian Hahner, Emmanuel Letier, Marin Litoiu, Lina Marsso, Angelika Musil, Juergen Musil, Genaína Nunes Rodrigues, Diego Perez-Palacin, Federico Quin, Patrizia Scandurra, Antonio Vallecillo, and Andrea Zisman. 2023. Towards a Research Agenda for Understanding and Managing Uncertainty in Self-Adaptive Systems. *ACM SIGSOFT Softw. Eng. Notes* 48, 4 (2023), 20–36. https://doi.org/10.1145/3617946.3617951

[32] Danny Weyns, Ilias Gerostathopoulos, Nadeem Abbas, Jesper Andersson, Stefan Biffl, Premek Brada, Tomas Bures, Amleto Di Salle, Matthias Galster, Patricia Lago, et al. 2023. Self-Adaptation in Industry: A Survey. *ACM Transactions on Autonomous and Adaptive Systems* 18, 2 (2023), 1–44.

[33] Danny Weyns, Ilias Gerostathopoulos, Nadeem Abbas, Jesper Andersson, Stefan Biffl, Premek Brada, Tomas Bures, Amleto Di Salle, Patricia Lago, Angelika Musil, et al. 2022. Preliminary results of a survey on the use of self-adaptation in industry. In *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 70–76.

[34] Danny Weyns and Usman M. Iftikhar. 2023. ActivFORMS: A Formally Founded Model-Based Approach to Engineer Self-Adaptive Systems. *ACM Trans. Softw. Eng. Methodol.* 32, 1, Article 12 (feb 2023), 48 pages. https://doi.org/10.1145/3522585

[35] Jon Whittle, Peter Sawyer, Nelly Bencomo, Betty H. C. Cheng, and Jean-Michel Bruel. 2010. RELAX: a language to address uncertainty in self-adaptive systems requirement. *Requir. Eng.* 15, 2 (2010), 177–196. https://doi.org/10.1007/S00766-010-0101-0