Article:
Carwehl, Marc, Imrie, Calum Corrie, Vogel, Thomas et al. (2026) Parley+: Uncertainty Reduction in Self-Adaptive Systems. ACM Transactions on Autonomous and Adaptive Systems. 25. ISSN: 1556-4703

https://doi.org/10.1145/3746234

PARLEY+: Uncertainty Reduction in Self-Adaptive Systems

MARC CARWEHL, Institut für Informatik, Humboldt-Universität zu Berlin, Berlin, Germany
CALUM IMRIE, Department of Computer Science, University of York, York, UK
THOMAS VOGEL, Institut für Informatik, Humboldt-Universität zu Berlin, Berlin, Germany
GENAINA RODRIGUES, Department of Computer Science, University of Brasilia, Brasilia, Brazil
RADU CALINESCU, Department of Computer Science, University of York, York, UK
LARS GRUNSKE, Institut für Informatik, Humboldt-Universität zu Berlin, Germany

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 previous work proposed PARLEY, a more nuanced, adaptive approach to SAS uncertainty reduction. To that end, we introduced an 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 tradeoffs between SAS uncertainty reduction benefits and new information acquisition costs with guarantees for the satisfaction of requirements. In this article, we extend PARLEY to PARLEY+ by improving the synthesis of these controllers and by expanding the formalisation of PARLEY+ to prove the validity of the synthesis. We illustrate the use and extend the evaluation of the effectiveness of our approach for mobile robot navigation and service-based system SAS. The evaluation results show that PARLEY+ can synthesise controllers that help achieve the system's objectives significantly better than PARLEY in 88.1% of the cases.

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

Additional Key Words and Phrases: controller synthesis, uncertainty, self-adaptive systems

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).

Authors' Contact Information: Marc Carwehl (corresponding author), Institut für Informatik, Humboldt-Universität zu Berlin, Berlin, Germany; e-mail: m.carwehl@hu-berlin.de; Calum Imrie, Department of Computer Science, University of York, York, UK; e-mail: calum.imrie@york.ac.uk; Thomas Vogel, Institut für Informatik, Humboldt-Universität zu Berlin, Berlin, Germany; e-mail: thomas.vogel@informatik.hu-berlin.de; Genaina Rodrigues, Department of Computer Science, University of Brasilia, Brasilia, Brazil; e-mail: genaina@unb.br; Radu Calinescu, Department of Computer Science, University of York, York, UK; e-mail: radu.calinescu@york.ac.uk; Lars Grunske, Institut für Informatik, Humboldt-Universität zu Berlin, Germany; e-mail: grunske@informatik.hu-berlin.de.

ACM 1556-4703/2026/6-ART25
https://doi.org/10.1145/3746234

ACM Reference format:
Marc Carwehl, Calum Imrie, Thomas Vogel, Genaina Rodrigues, Radu Calinescu, and Lars Grunske. 2026. Parley+: Uncertainty Reduction in Self-Adaptive Systems. *ACM Trans. Autonom. Adapt. Syst.* 21, 3, Article 25 (June 2026), 35 pages.
https://doi.org/10.1145/3746234

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 (CPS)** 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* [16, 34]. 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. [25]) control loops are termed **Self-Adaptive Systems (SAS)**.

The growing demand for SAS in many application domains [47, 48] has led to the development of numerous self-adaptation solutions over the past two decades. Nevertheless, most 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 article, 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 [15].

As a first example, consider the UNDERSEA exemplar [18]. 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 **Tele-Assistance Service (TAS)** exemplar [45]. 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's 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 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 reductions, 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 and so on.

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 and so on. 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 article is to determine *which* **Uncertainty Reduction Services (URSs)** *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 SAS*, which we call Parley[2]. 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 (Figure 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 URSs 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 in our Parley architecture has two major benefits. First, as with any architecture that promotes 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 URC straightforward. In this article, we explore the latter.

In addition to this new architecture, Parley comes with a tool-supported method for the formal synthesis of URCs 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 meta-heuristic search to synthesise URCs guaranteed to satisfy strict reliability, performance and other quality constraints, and to be Pareto-optimal concerning a set of quality optimisation objectives.

This article is an extension of our SEAMS 2024 paper [12] that presented Parley with the following contributions:

— The Parley hybrid self-adaptation paradigm and associated two-tier controller architecture;
— The Parley method for synthesising correct-by-construction URCs;

[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.

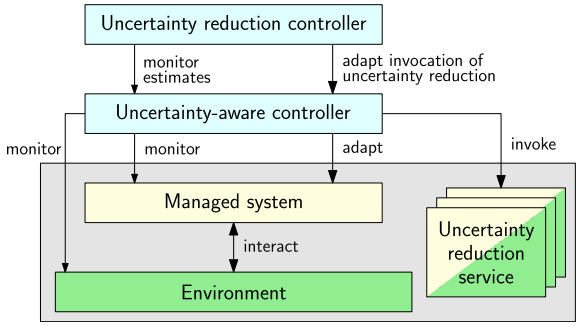


Fig. 1. Parley+ SAS architecture: an uncertainty reduction controller drives the adaptive reduction of epistemic uncertainty through the invocation of uncertainty reduction 'services' provided by the managed system, its environment or a combination thereof. URC, uncertainty reduction controller.

- A toolchain which automates the application of the URC synthesis method and which includes a new tool for augmenting the DTMC 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.

We present Parley+ in this article, which conceptually extends Parley with an improved initialisation of the search for optimal URCs. While Parley uses random policies to start the search, Parley+ uses additionally policies that invoke URSs at fixed time intervals. These policies provide a promising starting point for the search for optimal URCs as they are distributed in the search space. With this extension, Parley+ provides URCs that typically outperform Parley's URCs in our case studies. Moreover, we conceptually extend the formalisation of Parley+ to prove the validity of our approach concerning the synthesis of URCs (i.e., the model augmentation). Particularly, we prove that the model augmentation in Parley+ produces a valid model that can simulate any behaviour of the original model. Proposing Parley+, we further extend the evaluation to pairwise compare Parley+, Parley and a baseline and increase the number of evaluation settings from 9 to 16 for a thorough comparison. In this evaluation, we replaced the 'server infrastructure management' case study with TAS being more complex in terms of states and possible URCs to pose a more realistic problem to Parley+. Additionally, this case study demonstrates Parley+'s ability to use *multiple* URSs for the first time, despite that our formalisation already covered this aspect. Finally, we extend our evaluation of Parley+'s scalability by conducting more experiments to obtain more confidence in the results. In summary, this article provides the following novel contributions:

- Improved synthesis of optimal URCs that typically outperform Parley's URCs;
- Formalisation of Parley+ proving the validity of the URC synthesis;
- Extended evaluation of Parley+ against Parley and a baseline with more evaluation settings;
- Novel case study (TAS) that demonstrates the use of multiple URSs for the first time;
- Extended evaluation of Parley+'s scalability with more experiments;
- Adding more competing approaches in the discussion of related work to better position Parley+.

Article Structure. The rest of the article is organised as follows. In Section 2, we compare Parley+ to related research on SAS. Next, Section 3 provides a running example used to illustrate the components and stages of Parley+ in its description from Section 4. Finally, Section 5 presents our evaluation of Parley+, and Section 6 concludes the article with a summary and suggestions for further work.

2 Related Work

Handling uncertainty has been one of the driving forces of research in the area of SAS [10, 22, 34, 40, 46, 50]. In the following, we review selected approaches that are particularly related to PARLEY+. Uncertainty handling may follow a control-theoretical [26, 36, 43, 44], an architecture-based [13, 35, 37, 38, 49] or a planning-based [1, 9, 16, 35, 38, 41] adaptation approach.

Control-Theoretical Approaches. Shevtsov et al. [43] apply control theory to dealwith 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. [36] developed a framework for evaluating the safety of autonomous driving using end-to-end Bayesian Neural Network 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 controllers aim for a broader spectrum where epistemic uncertainties can be mitigated as the system objectives are satisfied. The work by Kobayashi et al. [26] 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. [44] 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. **Active Monitoring Mechanism (AMM)** [39] proposes a stochastic framework to systematically evaluate the confidence that a model deviation has occurred. While AMM is about actively monitoring the system in real-time and adapting based on observed conditions, PARLEY+ is about formally synthesising controllers at design-time that reduce uncertainty through anticipation rather than reaction. González et al. [23] introduce a control strategy that generates multiple scenarios based on potential future states and selects the best action based on these scenarios. While offering real-time adaptability, this work relies heavily on the quality of its predictive model, which can introduce limitations if the model does not accurately capture the dynamics of the system. PARLEY+ does not rely on an MPC model and rather synthesises controllers based on formal specifications. VERACITY [2] integrates confidence-interval quantitative verification with an adaptive uncertainty reduction heuristic that collects additional data about the parameters of the verified model by unit-testing specific system components over a series of verification iterations. While VERACITY focuses on the domain of component and unit testing, PARLEY+ goes an abstraction level higher so that the controller can suit a broader adaptation domain.

> From the control-theoretical perspective, PARLEY+ differs from existing approaches by synthesising URCs that follow any formalised system objectives. To this end, PARLEY+ only relies on a formal DTMC model of the existing system and environment. With the URC, we provide a clear separation of concerns within the controllers. Additionally, we pre-compute the URC's actions at design time to accommodate resource constriction at runtime.

Architecture-Based Adaptation Approaches. Regarding the handling of uncertainties following an architecture-based adaptation approach, Weyns and Iftikar [49] propose ActivFORMS-ta, an

architecture-based adaptation approach based on the MAPE-K reference model. Moreno et al. [37] introduced the concept of uncertainty reduction to manage uncertainty in SAS and show how uncertainty reduction decisions can be integrated into self-adaptation decisions. Our work addresses some challenges put forward by Moreno et al. Parley+ focuses on allowing an explicit representation and the correct-by-construction synthesis of URCs. In particular, employing the estimate of the UAC, Parley+ can act on handling uncertainties if and when necessary. We believe the work by Camara et al. [13] 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 URS in the formalism of **Parametric DTMC (pDTMC**). Additionally, Parley+ resorts to a meta-heuristic approach to find near-optimal adaptation policies. Kreutz et al. [29] 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. Caldas et al. [8] propose the design of a PI controller following separations of concerns to improve adaptability in uncertain and dynamic conditions. They integrate traditional control strategies of PI controllers and machine learning. However, their focus is mostly on SASO properties other than a more generic approach as provided by Parley+. In addition, Parley+ provides a more structured, proactive approach to controller synthesis, ensuring robustness and reliability, particularly in critical systems given its formal guarantees.

From the architecture-based perspective, Parley+ allows for an explicit representation of uncertainty through the notion of estimates. In particular, employing the estimate of the UAC, Parley+ can act on handling uncertainties if and when necessary. Moreover, Parley+ takes probabilities into account and provides means to build extensible controllers and relies on meta-heuristic search to provide adequate controllers even when traditional synthesis approaches struggle with scalability.

Planning-Based Approaches. Various planning-based approaches for tackling the decision-making process under uncertainties have also been proposed in the literature [1, 9, 16, 35, 38, 41]. **Partially Observable Markov Decision Processes (POMDP)** have been extensively used: (i) in the robotic domain to reason for imperfect robot actions and environment observations [31], (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 [38], (iv) the human-robot uncertainty interaction problem [35] or (v) on the decision-making process for SASs while offering awareness of **Non-Functional Requirements (NFRs)** priorities at runtime through a vector-valued reward function [41]. 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 behaviour and resolving (epistemic) uncertainty in the URC. Esfahani et al. [16] 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 tradeoffs, 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. Garcia et al. [17] present a probabilistic approach to quantify the uncertainty associated with the effects of adaptation actions on the state of a SAS supported by Bayesian inference and POMDPs. These effects

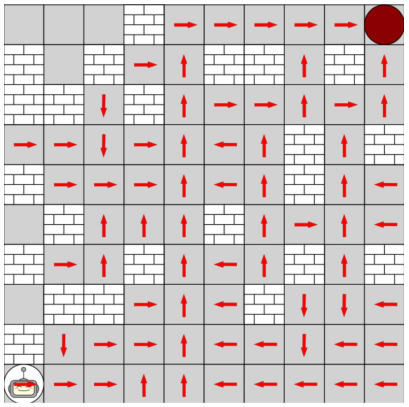


Fig. 2. A sample map for our running example. The robot starts in the lower left corner and its mission is to traverse to the destination in the upper right corner without crashing into static obstacles. Red arrows denote move commands.

are translated into the satisfaction levels of the NFRs to, therefore, drive the decision-making. While that work utilises the formal models at runtime, PARLEY+ uses its formal model at design time to provide controllers, reducing the overhead at runtime. Camilli et al. [11] propose an approach that tames uncertainty estimate via Bayesian Model Averaging by averaging predictions from multiple models, with the idea that multiple models reflect uncertainty. PARLEY+, on the other hand, includes such estimates suffering uncertainty into a single model, providing the possibility to use formal methods such as model checking before runtime. Sánchez Salas et al. [42] recently proposed an approach to deal with uncertainty interaction from temporal availability constraints and element reliability. While PARLEY+ does not explicitly discuss uncertainty interaction, our approach allows the underlying model to invoke this concept and use the advantages presented in [42] and similar works.

Using DTMCs, PARLEY+ provides a more efficient method to synthesise controllers than approaches relying on POMDPs. Additionally, PARLEY+ allows to represent uncertainty-affected estimates in one model, with a logic to describe how the estimates change during the system's execution. Finally, PARLEY+ also allows to model uncertainty interaction.

3 Running Example

Throughout this article, we use a simplified discrete mobile robot as a running example, navigating within the constraints of a known 10×10 discrete grid map. Maps of similar size have been investigated in literature [21]. While this example is naive in describing the robot's navigation, it serves as a suitable demonstrator for our approach. 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

adapts moves based on its estimate of the robot's location $(\hat{x}, \hat{y})$. Since the focus of this article is on uncertainty reduction, we simplify the self-adaptation controlling the robot's movement. That is, the movement controller adapts the direction of the robot's movement based on the environment represented by the robot's estimated location and nearby obstacles. This results in a system behaviour, in which the robot is moving in one direction until the controller decides to adapt the direction based on the estimated location and obstacles.

We consider that each move can have uncertain outcomes, for instance, due to wheel slip [14]. 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 use the estimated location and determine a threshold that depicts after how many moves the localisation service should be used at the estimated location to minimise cost while maximising the mission success rate. A policy for setting this threshold can be modelled as a function $f(\hat{x}, \hat{y}) = \bar{c}$ where $(\hat{x}, \hat{y})$ represents the estimated location of the robot and $\bar{c}$ denotes the threshold 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 when the localisation service should ideally be invoked to minimise mission costs while maximising the mission success rate.

4 Uncertainty Reduction with 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 the enactment of a controller. 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.

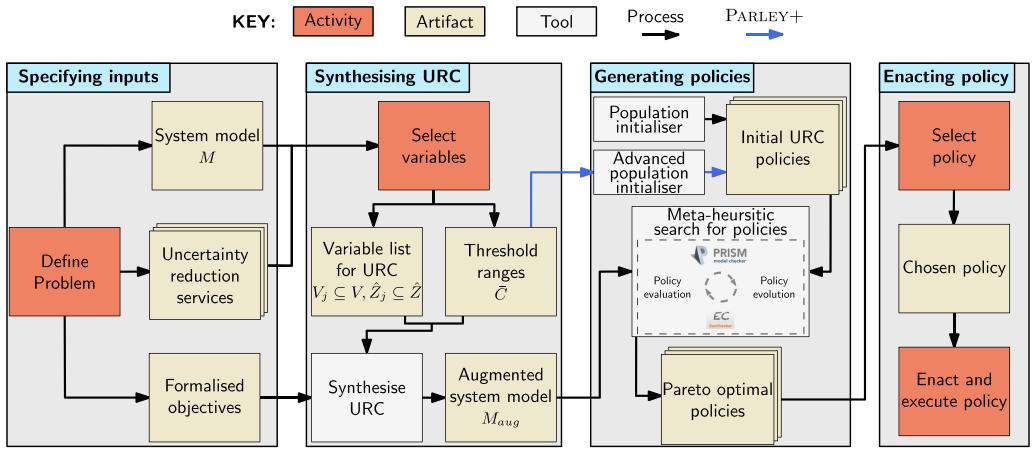


Fig. 3. Process for PARLEY (black arrows) and PARLEY+ (black and blue arrows).

In PARLEY+, the existing notion of a *change management layer* (cf. [28]) is handled by two separate controllers, as visualised in Figure 1: First, an 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 URC is solely concerned with mitigating epistemic uncertainty to aid the UAC in its decision-making by adapting when and how the UAC invokes the URSs to acquire new information.

> With our two-layer control architecture, we propose a separation of concerns, with the UAC reducing uncertainty in its estimates at a fixed rate and the URC dedicated to adapting this rate 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 after a fixed number of moves (threshold). The URC adapts the threshold with which the localisation service is invoked to reduce uncertainty based on the estimated location.

4.2 Process

The remainder of this section discusses the PARLEY+ methodology step-by-step. Figure 3 visualises these steps from input specification to the controller synthesis and URC-policy generation until the enactment of a selected policy within the URC. Moreover, Figure 3 visualises the differences between PARLEY+ and PARLEY in terms of the population initialiser and behavioural flow (blue arrows).

4.2.1 Specifying Inputs. Initially, given a problem definition, the stakeholders specify a system model M, which captures the necessary and relevant aspects of the SAS and environment, including a list of URSs. In addition, they specify objectives. PARLEY+ does not provide automated techniques for creating the specification, and we refer to work on mining behavioural models such as [20, 33] to semi-automate these steps.

First, we begin by clarifying assumptions about the system model M that depicts the managed system, the UAC, and the environment. We only require that these three aspects are modelled as a DTMC M. Consequently, different kinds of domain and adaptation logics can be used in the managed system and UAC. For instance, the adaptation logic can be realised by adaptation rules or

online learning. When creating M, we assume that the behaviour of the system in its environment has been appropriately captured and will not deviate drastically at runtime. If a substantial shift of the system or environment compared to M occurs at runtime, M will have to be updated and the PARLEY+ approach will have to be re-run at runtime, which is out of the scope of this article. Here, we focus on specifying M and how PARLEY+ is applied to it.

M consists of states, one of them being the initial state, and a transition matrix P that depicts probabilities of transitioning from one state to another. Each state in M is a tuple:

$$s = (V, \hat{Z}, Z, C, \bar{C}),$$

where V corresponds to variables whose information is fully known to the UAC, and $\hat{Z} \in \hat{\zeta}$ is the UAC's estimate of the information $Z \in \zeta$ that the UAC does not know. Note that V, $\hat{Z}$ and Z typically depict multiple variables and should therefore be considered as tuples, e.g., $V = (v_1, \ldots, v_n)$. The tuples $\hat{Z}$ and Z have the same size and any $\hat{z}_i \in \hat{Z}$[3] is the UAC's estimate of $z_i \in Z$. Due to the uncertainty, the UAC knows only V and $\hat{Z}$ and its adaptation decisions are solely based on V and $\hat{Z}$. The UAC's estimates $\hat{Z}$ can be optimised by using URSs. We assume that the stakeholders specify at least one such service and that each service uses at least one aspect of the ground truth to improve at least one estimate. Formally, each of the w URSs is modelled as a function. To this end, the jth ($j \in [1, w]$) URS is defined over a subset of the ground truth $Z_j \in \zeta_j$. We use $\pi_j(\zeta)$ as *projection* to provide all tuples along the dimensions specified in j as $\zeta_j = \pi_j(\zeta)$. The URS then provides better estimates $\hat{Z}_j \in \hat{\zeta}_j = \pi_j(\hat{\zeta})$:

$$urs_j : \zeta_j \rightarrow \hat{\zeta}_j.$$

Thus, a service updates at least one of the UAC's estimates $\hat{Z}_j \sqsubseteq \hat{Z}$ using the ground truth $Z_j \sqsubseteq Z$. Which estimates are updated depends on the capabilities of a service such as which aspects of the ground truth can be obtained by the service. After an update, the UAC's estimates are aligned with the ground truth but start deteriorating momentarily due to uncertainty. Therefore, repetitive updates of the estimates are required whenever uncertainty accumulates and causes deviations between the estimates and ground truth. Besides epistemic uncertainty in the system, each service might suffer from aleatoric uncertainty, which can be modelled within the corresponding function, accordingly, for instance, by adding noise to the ground truth.

The UAC invokes each service naively after a fixed number of steps in the model. We assume that thresholds depicting these fixed numbers of steps are encoded in a constant tuple, namely $\bar{C}$ of size w, where w is the number of URSs, such that $\bar{c}_j \in \bar{C}$ is the interval between two invocations of service urs_j and $\frac{1}{\bar{c}_j}$, therefore, its frequency. To this end, C depicts w counters, modelling that a specific URS j is to be invoked if the counter for the service $c_j \in C$ exceeds its threshold $\bar{c}_j \in \bar{C}$.

The system's behaviour is defined by transitions over the system's states with a transition probability matrix P. Each transition t has a source state s and a target state s'. P assigns a probability p_t to each transition t:

$$P(t) = P(s, s') = P((V, \hat{Z}, Z, C, \bar{C}), (V', \hat{Z}', Z', C', \bar{C}')) = p_t.$$

We assume that any transition that has a non-zero probability in P falls into one of the following two categories:

[3]In this context, the notation $A_j \sqsubseteq A$ refers to sub-tuples of the tuple A. Specifically, A is an ordered tuple, and A_j represents a sub-tuple of A consisting of selected elements from A, preserving their order. Additionally, the notation $A \setminus A_j$ represents the tuple obtained by removing the elements of the sub-tuple A_j from A, while maintaining the order of the remaining elements and $a_i \in A$ is used to indicate that a_i is the ith element of the tuple A. This differs from the standard set-theoretic interpretation, as we are working with ordered structures.

(1) *Behaviour Transitions.* Transitions that model behaviour in the environment, UAC and managed system. These transitions affect the UAC, managed system and the environment, including the UAC's estimates thereof. Thus, any such transition might alter V, $\hat{Z}$ and Z. Moreover, each transition t is associated with an increment function $\oplus_t$ that increments selected counters $c_i \in C$ whenever the transition is taken (cf. Equation (1b)); counters that are not incremented remain unchanged. Notably, the thresholds $\bar{C}$ remain unchanged by this category of transitions (cf. Equation (1c)). A *behaviour transition* can only be taken when no counter exceeds its threshold (cf. Equation (1a)). Formally, such a *behaviour transition* is a tuple with the corresponding constraints:

$$((V, \hat{Z}, Z, C, \bar{C}), (V', \hat{Z}', Z', C', \bar{C}')) \text{ with} \tag{1}$$

$$\forall c_i \in C, \bar{c}_i \in \bar{C} : c_i \leq \bar{c}_i, \tag{1a}$$

$$C' = C \oplus_t 1 \text{ and} \tag{1b}$$

$$\bar{C}' = \bar{C}. \tag{1c}$$

(2) *Uncertainty Reduction Transitions.* Transitions that model invocations of an URS by the UAC to reduce uncertainty in the UAC's estimates. With one such transition, exactly one service is invoked. For instance, for an invocation of URS j, the transition goes from $s = (V, \hat{Z}, Z, C, \bar{C})$ to $s' = (V', \hat{Z}', Z', C', \bar{C}')$ (cf. Equation (2)) and is enabled only if the corresponding counter c_j exceeds the threshold $\bar{c}_j$ (cf. Equation (2a)). When taking the transition, URS j updates the estimates $\hat{Z}_j$ to $\hat{Z}'_j$ and the remaining estimates $\hat{Z} \setminus \hat{Z}_j$ may be changed arbitrarily (cf. Equation (2b)). We also do not impose any restriction on V' or Z'. Additionally, the counter for the invoked service is reset ($c'_j = 1$, cf. Equation (2c)). Notably, similar to the *behaviour transitions*, the thresholds $\bar{C}$ remain unchanged by this category of transitions (cf. Equation (2d)):

$$((V, \hat{Z}, Z, C, \bar{C}), (V', \hat{Z}', Z', C', \bar{C}')) \text{ with} \tag{2}$$

$$c_j \in C, \bar{c}_j \in \bar{C} : c_j > \bar{c}_j, \tag{2a}$$

$$\hat{Z}'_j = urs_j(Z_j) \text{ where } \hat{Z}'_j \subseteq \hat{Z}' \text{ and } Z_j \subseteq Z, \tag{2b}$$

$$C' = (c_1, \ldots, c'_j, \ldots, c_w) \text{ with } c'_j = 1 \text{ and} \tag{2c}$$

$$\bar{C}' = \bar{C}. \tag{2d}$$

With this formalisation, we allow for an invocation of a URS to consume system resources or affect the environment, which is represented in the knowledge and can be changed without restriction by the uncertainty reduction transition (i.e., V', $\hat{Z}' \setminus \hat{Z}'_j$, Z').

Note that the *behaviour transitions* and *uncertainty reduction transitions* have mutually exclusive pre-conditions on the counters C and thresholds $\bar{C}$ (cf. Equations (1a) and (2a)). That is, as long as no counter exceeds its threshold, only *behaviour transitions* are taken but not *uncertainty reduction transitions*. On the other hand, as long as at least one counter exceeds its threshold, no *behaviour transition* is taken but only *uncertainty reduction transitions*.

Lastly, we assume that the stakeholders will formalise 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 SAS, with a controller (UAC) relying on its estimate of variables suffering uncertainty to adapt the managed system. The UAC invokes URSs to reduce the uncertainty in its estimates naively using *static* thresholds $\bar{C}$ and counters C. We assume that the system, including its UAC, managed system and environment, as well as the URSs are modelled as a DTMC M.

Example. For our robotic example, we construct a DTMC with PRISM [30] (cf. Listing 1). We employ PRISM's notion of modules which synchronise over labelled transitions, i.e., the transition *east*, cf. ll. 8, 17 and 28, can be invoked only if all three modules can invoke the transition, invoking it in one single step and accumulating 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 Section 3. This module (cf. ll. 16–20) controls where the robot should move depending on its estimated location $\hat{Z} = (\hat{x}, \hat{y})$, as depicted, e.g., in l. 17. Any of the robot's moves is considered a *behaviour transition*. We modelled the UAC in this way as it is a pragmatic abstraction of the robot's adaptive movements based on the robot's estimated location. For simplicity, we model that the UAC adapts the robot's movement after every step (i.e., with change of the estimated location) although in practice, the robot might move independently from the controller and its movements are only adapted whenever necessary, for instance, when the robot is close to an obstacle. Finally, the controller's knowledge is handled by a dedicated module that includes the estimated location, cf. ll. 24 and 25. Additionally, a counter $C = (c_1)$ 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. Section 3) that aligns the estimated location with the actual location. It is invoked if c_1 exceeds a constant limit $\bar{c}_1 \in \bar{C}$ (cf. l. 22), which may have been set to, e.g., $\bar{c}_1 = 2$. c_1 is reset (cf. transition *localisation*, l. 1), accordingly. In this sense, the *localisation* transition is an *uncertainty reduction transition*. Note, that any movement or invocation of the localisation service incurs a cost (cf. ll. 35–40).

4.2.2 Synthesising the URC. We propose to synthesise a dedicated controller, the URC, to control when the UAC employs its URSs. To this end, the URC adapts the thresholds $\bar{C}$. The URC performs these adaptations based on the UAC's knowledge represented in the variables V and $\hat{Z}$. We propose that the designers select which of these variables the URC should use to decide how it should adapt the UAC[4], as well as select the threshold ranges for $\bar{C}$, cf. *Select Variables* in Figure 3. This selection depends on the system's design space, e.g., which information can be monitored [7]. The stakeholders have to take domain knowledge into consideration to make suitable choices for variables that represent the system state pragmatically regarding the adaptation problem and bounds of thresholds that are sensible for the system. Note, that adaptation decisions cannot be made based on the ground truth Z as the ground truth is unknown to both controllers. We depict possible adaptation decisions as a *policy* based on the selected set of variables as a function:

$$decision : \Lambda \times \hat{\zeta} \rightarrow \bar{\Gamma},$$

for any value in the selected subset of $V \in \Lambda$ and $\hat{Z} \in \hat{\zeta}$, accordingly, that computes the thresholds $\bar{C} \in \bar{\Gamma}$ for all URSs. To synthesise the URC, we augment the DTMC M such that $\bar{C}$ can be adapted dynamically in M, corresponding to the desired thresholds. Therefore, an additional, third category of transitions with non-zero probability is added to M:

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

```
dtmc
const int N = 9; //map size
const double p = 0.01; // probability of deviation in moves

module Robot
  x : [0..N] init 0;
  y : [0..N] init 0;
  [east] true ->
    (1-3*p): (x'=min(x+1, N)) +
    p: (y'=min(y+1, N)) +
    p: (y'=max(y-1, 0)) +
    p: (x'=max(x-1, 0));
  ...
endmodule

module Adaptation_MAPE_Controller
  [east]  (x̂=0) & (ŷ=0) -> true;
  [north] (x̂=1) & (ŷ=0) -> true;
  ...
endmodule

const int c̄1 = 2;
module Knowledge
  x̂ : [0..N] init 0; //estimated position
  ŷ : [0..N] init 0; //estimated position
  c1 : [1..10] init 1;
  ready : Bool init true;
  [east]  ready -> (x̂'=min(x̂+1, N)) & (ready'=false);
  [north] ready -> (ŷ'=min(ŷ+1, N)) & (ready'=false);
  ...
  [localisation] c1>c̄1 & !ready -> (x̂'=x) & (ŷ'=y) & (c1'=1) & (ready'=true);
  [skip] c1 ≤ c̄1 & !ready -> (c1'=c1+1) & (ready'=true);
endmodule

rewards "cost"
  [east] true : 1;
  [north] true : 1;
  ...
  [localisation] true : 5;
endrewards
```

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

(3) Threshold Adaptation Transitions. Transitions that adapt when URSs are invoked by changing the thresholds. These transitions solely adapt the threshold values $\bar{C}$ (cf. Equation (3e)). No other variables are changed by this transition (cf. Equations (3a)–(3d)) because the UAC can only be adapted using the thresholds $\bar{C}$. Keeping a clear separation between transitions of the system model M (cf. previously defined *behaviour transitions* and *uncertainty reduction transitions*) and *threshold adaptation transitions* provides a clear separation of concerns:

$$((V, \hat{Z}, Z, C, \bar{C}), (V', \hat{Z}', Z', C', \bar{C}')) \text{ with} \tag{3}$$

$$V' = V, \tag{3a}$$

$$\hat{Z}' = \hat{Z}, \tag{3b}$$

$$Z' = Z, \tag{3c}$$

$$C' = C \text{ and} \tag{3d}$$

$$\bar{C}' = decision(V, \hat{Z}). \tag{3e}$$

Adding *threshold adaptation transitions* to the system model M containing *behaviour and uncertainty reduction transitions* results in an augmented model M_{aug}. The augmented transition matrix P_{aug} of M_{aug} resembles all behaviour from the original model M with the transition probability matrix P as well as the URC, and is defined as follows:

$$P_{aug}(s, s') = P_{aug}((V, \hat{Z}, Z, C, \bar{C}), (V', \hat{Z}', Z', C', \bar{C}')) = \tag{4}$$

$$\begin{cases} P(s, s') & \text{if } decision(V, \hat{Z}) = \bar{C} \text{ (behaviour or uncertainty reductions transition as defined in Eqs. (1) and (2))} \quad (4a) \\ 1 & \text{if } decision(V, \hat{Z}) = \bar{C}' \neq \bar{C} \wedge V' = V \wedge Z' = Z \wedge C' = C \text{ (threshold adapt. tr. as defined by Eq. (3))} \quad (4b) \\ 0 & \text{if } decision(V, \hat{Z}) \neq \bar{C} \wedge decision(V, \hat{Z}) \neq \bar{C}' \text{ (others).} \quad (4c) \end{cases}$$

If the URC decides not to adapt the thresholds, i.e., the outcome of the decision function matches the current thresholds, the augmented model M_{aug} behaves as the system model M. In this case, only *behaviour* and *uncertainty reduction transitions* can be taken and have the same probability as in M (cf. Equation (4a)). In contrast, when the URC decides to adapt the thresholds, i.e., the outcome of the decision function does not match the current thresholds, then the adaptation must happen. Thus, a *threshold adaptation transition* must happen (probability of the transition is 1) that adapts the current thresholds according to the decision function (cf. Equation (4b)). In this case, the *behaviour* and *uncertainty reduction transitions* have a probability of 0. Any other transition that would adapt the thresholds in another way has probability 0 (cf. Equation (4c)).

From a practical point of view, our implementation using PRISM models allows designers to prevent *threshold adaptation transitions* from taking place in certain states despite the URC's decision that an adaptation should take place. The reason for preventing *threshold adaptation transitions* is to allow a sequence of *behaviour and uncertainty reduction transitions* to be executed as an atomic transaction. In other words, we allow the URC to adapt the UAC only when the system and UAC are in a *quiescent* state [27], that is, no transactions are currently running. We do not include the notion of transactions and quiescence into our formalisation since any transaction comprising multiple actions of the system can be formalised as a single *behaviour transition* and any transaction comprising invocations of multiple URSs can be formalised as a single *uncertainty reduction transition*. In this case, quiescence can be achieved by construction because each transition is considered a transaction and a threshold adaptation only occurs in-between completed transactions.

Since the *decision* function has not been implemented yet, the concluding model M_{aug} becomes a pDTMC. The parameters implement the *decision* function in the pDTMC. Concrete values for these parameters directly serve as the result of the function, that is, the desired threshold values $\bar{C}$, such that one parameter $param_{V,\hat{Z}}$ reflects the output of the decision function $decision(V, \hat{Z})$. An assignment of a concrete value to every parameter is called a *policy*. Usually, multiple policies exist for a pDTMC, implementing different adaptation logics for the URC.

We synthesise a URC that adapts the thresholds with which URSs are invoked by the UAC. The adaptation decisions are modelled as parameters, resulting in a pDTMC.

Example. In our example, we automatically add a module *Uncertainty_Reduction_Controller* (cf. ll. 5–14 in Listing 2) to the model and move the variable $\bar{c}_1$ to this module (cf. l. 6 in Listing 2)[5]. Hence, the URC can adapt $\bar{c}_1$. To depict quiescent states in which an adaptation is permitted, we use *turn* (cf. l. 7 in Listing 2) that sequentially interrupts the managed system and UAC to allow for adaptations performed by the URC, i.e., when *turn* = 2 (cf. l. 10). 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

[5]L. 26 in Listing 1 is deleted, accordingly.

```
...
const int decision_0_0; //adapted threshold for invoking the uncertainty reduction service at estimated location (0, 0)
const int decision_0_1; //adapted threshold for invoking the uncertainty reduction service at estimated location (0, 1)
...
module Uncertainty_Reduction_Controller
  c̄1 : [1..10] init decision_0_0;
  turn : [1..3] init 1;
  [east] turn=1 -> (turn'=2);
  ...
  [] turn=2 & x̂=0 & ŷ=0 -> (c̄1'=decision_0_0) & (turn'=3);
  ...
  [localisation] turn=3 -> (turn'=1);
  [skip] turn=3 -> (turn'=1);
endmodule
```

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

$\hat{y}$, 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.

Theorems and Proofs. With this step, we augmented the system DTMC M to a pDTMC M_{aug}. We formulate the following theorem, expressing the validity of M_{aug}:

THEOREM 1. *If M is a valid DTMC, the augmented model M_{aug} is a valid pDTMC.*

Following the definition of a DTMC in the *Principles of Model Checking* [4, p. 747], to prove that M_{aug} is a valid pDTMC, we need to argue that for any selection of parameters:

(1) the number of states in M_{aug} is greater than zero and countable,
(2) the initial distribution (depicting initial states) sums to 1 and
(3) for each state, the sum of probabilities of outgoing transitions is 1.

PROOF. M_{aug} has as many states as M since we neither add nor remove states in the augmentation process. Therefore, if M has only countably many states (definition of a DTMC), then the same holds for M_{aug}. Similarly, we argue that M_{aug} has at least one state as long as M has at least one state.

Since we do not change the initial state (or the initial distribution, to be more general), the second condition is satisfied by our augmented model.

Considering the probabilities of outgoing transitions, we argue that the transitions that are enabled for a state s in M_{aug} depend solely on s (memoryless property, cf. [4, p. 747]). We designed P_{aug} such that two cases are mutually exclusive for transitions enabled in s, cf. Equations (4a) and (4b). On the one hand, if in $s = (V, \hat{Z}, Z, C, \bar{C})$, where $decision(V, \hat{Z}) \neq \bar{C}$ then the *threshold adaptation transition* is enabled. By definition (cf. Equation (3)) for any given policy exactly one *threshold adaptation transition* exists for any state s. In P_{aug}, this transition receives probability 1 (cf. Equation (4a)). All behaviour and uncertainty reduction transitions are disabled (probabilities are 0) because their pre-condition $decision(V, \hat{Z}) = \bar{C}$ is violated. On the other hand, if $decision(V, \hat{Z}) = \bar{C}$ in s, any transition from the system model is enabled with the same probability as in M. The threshold adaptation transition has a probability of 0 because its pre-condition $decision(V, \hat{Z}) \neq \bar{C}$ is violated. Therefore, if M is a valid DTMC, then these probabilities sum to 1 for each state s in M, and by definition also in M_{aug}, regardless of the selected policy, that is, concrete values obtained by the decision function. □

Additionally, we claim that we can select the parameters in M_{aug} such that it simulates the system model M:

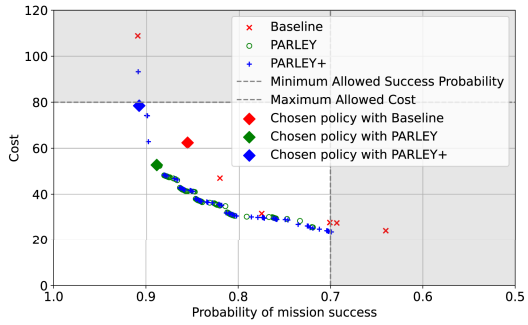
Theorem 2. *For any choice of static thresholds in M, there exists a policy such that M_{aug} simulates M, i.e., any trace in M, provided as a list of states $s_1, \ldots, s_n$, is matched by an identical trace with the same probability in M_{aug}.*

Proof. Given a tuple of thresholds $\bar{C}$, we can select the parameters in M_{aug} depicted by the *decision* function such that $decision(V, \hat{Z}) = \bar{C}$, i.e., we can use the *decision* function in M_{aug} to always set the thresholds as they are set in the original model (in other words, we do not adapt the thresholds). With such a choice of parameters, the condition $decision(V, \hat{Z}) \neq \bar{C}$ is never satisfied in any state in M_{aug}. Therefore, no *threshold adaptation transition* is enabled in the transition matrix P_{aug} and thus in M_{aug}. Thus, any transition in P_{aug} has the same probability as in P, i.e., $P(s, s') = p = P_{aug}(s, s')$. With this augmented model, exactly the same traces are possible in M_{aug} as in M, with the same probabilities. □

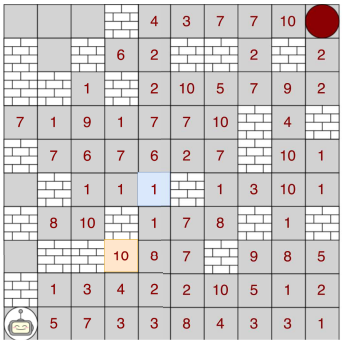
4.2.3 Generating Policies with Guarantees. In the previous step, we introduced the decision function $decision(V, \hat{Z})$ to trigger *threshold adaptation transitions* that determine the thresholds of invoking the URSs. In this step, we implement the decision function with concrete values assigned to the parameters of the pDTMC M_{aug}. As previously stated, assigning values to every parameter results in a *policy*, implementing one adaptation logic. Usually, multiple policies exist, spanning a search space of different adaptation logics for the URC. Thus, we define a search space for policies, which comprises any possible value for every parameter. The goal is to find suitable policies that satisfy or optimise the given system objectives. To assess the suitability of a policy, we evaluate if or how well a policy achieves the system objectives, obtaining the fitness of a policy. We compute the fitness of a policy using probabilistic model checking [30]. To this end, the objectives need to be formalised as **Probabilistic Computation Tree Logic (PCTL)** queries that are input into the model checker alongside the policy and the augmented system model M_{aug}. A query can formalise a probabilistic or quantitative objective. For a probabilistic objective, the model checker automatically computes the probability with which the policy satisfies the objective for M_{aug}. For a quantitative objective, the model checker computes the accumulated quantity (called *reward* [30]) achieved by the policy for M_{aug}. To compare policies with each other, we compare their fitnesses in terms of the probabilities and rewards of the objectives.

Usually, a system has more than one objective so that this corresponds to a multi-objective search/optimisation problem. Comparing two policies a and b can result in two succinct cases: a outperforms b in every objective, or a outperform b in some objectives while b outperforms a in other objectives. In the first case, we say that a Pareto-dominates b, resembling that a is better than b in every objective. Thus, a stakeholder would always choose a over b. In the second case, we say that a and b are Pareto-optimal policies and span a Pareto-front. Here, a stakeholder has to tradeoff between the objectives to eventually choose one policy of the Pareto-front.

Depending on the size of the search space of possible policies, an exhaustive search through the search space (i.e., evaluating all possible policies) might be infeasible. To this end, we propose to resort to a meta-heuristic search. Particularly, we use *EvoChecker* [19] that implements a meta-heuristic search through the search space of policies while relying on the PRISM model checker [30] to compute the fitness of policies. As a meta-heuristic search algorithm, the *Non-Dominated Sorting Genetic Algorithm II* is used. This algorithm follows the principles of evolution in biology and iteratively evolves a population of initially randomly generated individuals (in our case, policies) while optimising the fitness of the individuals. With each iteration (generation), new individuals are obtained by crossover and mutation of existing individuals. The fittest individuals are selected for the population of the subsequent generation. At the end of the search after a pre-defined number



(a) Pareto-front of policies in the baseline and generated by PARLEY+ and PARLEY. The stakeholders determine a maximum cost of 80 and select one policy, accordingly.



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

Fig. 4. Policy selection for the robotic example.

of generations, the algorithm ideally provides near-optimal individuals that are close to or even matching the globally optimal individuals.

Using meta-heuristic search, EvoChecker does not provide guarantees for global optimality of policies. However, invoking a probabilistic model checker such as PRISM [30] guarantees the satisfaction of the objectives and the resulting tradeoffs between objectives for each policy that has been identified by the meta-heuristic search.

These steps are represented in Figure 3 as *Generating policies*. In PARLEY, we rely on randomly generated initial URC policies (initial population) for the *Meta-heuristic search for policies*. PARLEY+ improves this search by using an *Advanced population initialiser*. This initialiser injects policies into the initial population, which invoke URSs with fixed rates. These policies provide a promising starting point for the search for near-optimal policies as they are distributed in the search space (see *Baseline* in Figure 4(a)). Formally, each of these policies can be described as a constant decision function: $\exists \bar{C}\ \forall V, \hat{Z} : decision(V, \hat{Z}) = \bar{C}$. That is, regardless of V and $\hat{Z}$, the same thresholds $\bar{C}$ are used. If necessary, the remaining individuals to fill up the initial population are randomly generated.

> Policies define values for the parameters and reflect the URC's behaviour. These policies can be evaluated by probabilistic model checking of the pDTMC. Using probabilistic model checking, we provide formal guarantees for each policy regarding the specified objectives. If the search space is too large, meta-heuristic search can be applied. Policies describing constant decision functions are used in the initial population of the meta-heuristic search.

Example. In our running example, each policy consists of 100 parameters each denoting the threshold for a possible (estimated) location on the map (100 cells in the 10×10 map). We choose to limit the maximum interval between two invocations of the localisation service to 10 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 be assigned any value in the interval between 1 and 10, the number of possible policies is 10^{100}. Due to the large search space, we use EvoChecker [19], a meta-heuristic approach to finding policies. In PARLEY+, we inject URC policies with static thresholds for uncertainty reduction in the initial URC policies (initial population) for the meta-heuristic search. To this end, we generate 10 policies, each depicting a constant threshold in the interval $\bar{C} \in \bar{\Gamma} = [1, 10]$.

The remaining 90 policies in the initial population are generated randomly. In contrast, PARLEY uses 100 randomly generated policies as initial URC policies.

To evaluate the fitness of a policy, we employ the objectives discussed in Section 3, accordingly: Objective $O1$ is maximising the probability that the robot reaches its destination (depicted through both x and y being 9). Objective $O2$ minimises the accumulated reward over the *cost* reward structure in the first 100 steps of a run ($C <= 100$), which is a reasonable upper bound in our model that is not reached in the mission. We formalise the objectives in PCTL using PRISM's specification language as follows:

$$O1 : Pmax =?(x = 9 \& y = 9),$$
$$O2 : Rmin\{"cost"\} =?[C <= 100].$$

Figure 4(a) visualises the Pareto-fronts of policies that PARLEY and PARLEY+ provide for the given map (cf. Figure 2) and guaranteed tradeoffs 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. The stakeholders tradeoff the objectives and select one of the policies of the Pareto-front accordingly for deployment. This tradeoff is essential as it enables the stakeholders to prioritise the objectives based on the requirements. Besides prioritisation of objectives by stakeholders, techniques to automatically select a policy can be used. One example of such a technique is the knee method [6].

Additionally, the stakeholders can impose constraints on the values utilised within the objectives. Such constraints limit the set of policies on the Pareto-front that are valid concerning the objectives, effectively eliminating policies that produce unacceptable probabilistic behaviours. For instance, a constraint might state that the probability of an accident in a safety-critical system should be below 0.1% or that on average the system should complete a task within 100 seconds. Accordingly, only policies that satisfy this constraint can be selected. If the constraints are too strict this can result in zero acceptable policies, the stakeholders will then need to consider whether to compromise, so to speak and weaken the constraints. Alternatively, the stakeholders might not want the constraints weakened, and instead redefine the problem with respect to the operational design domain and/or system for producing acceptable systems and policies.

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

Example. In our robotic example, the stakeholders encounter that the robot only has limited energy available during its mission. Therefore, they only consider policies that incur a cost of less than 80 as a constraint. Additionally, the stakeholders formulate that the probability of a successful mission should exceed 70%. Finally, they prioritise the success over the cost of a mission.

Figure 4(a) shows the Pareto-front of policies produced by various approaches. The white area depicts the subset of the Pareto-front that satisfies the stakeholders' constraints. Fortunately, the Pareto-front contains policies satisfying these constraints. Otherwise, the stakeholders would have to compromise on one of the constraints by weakening it. 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 usually accepted by the stakeholders.

In Figure 4(a), we use diamonds to highlight three policies depicting three policies obtained by PARLEY+ (blue), PARLEY (green) and the baseline, i.e., without a URC (red), that guarantee the highest mission success probability while staying below a cost of 80. We can see that PARLEY+ provides the best policy, i.e., the one with the highest probability of a successful mission for a cost below 80. The parameters in terms of the thresholds for localisation for each location that

are provided by one such policy are shown in Figure 4(b). Both the Pareto-front as well as every identified policy can be found in our publicly accessible online repository[6].

4.2.5 Enacting and Executing the Selected Policy. The policy selected in the previous step defines the adaptation logic of the URC. Thus, the URC implements the policy as follows (see Figure 1). At runtime, the URC monitors the variables chosen for the decision-making (selected from V and $\hat{Z}$), following the definition of the *decision function* (see Section 4.2.2). For every possible value of these variables, a policy prescribes the thresholds $\bar{C}$ that should be used by the system. That is, based on the monitored information, the URC adapts the thresholds $\bar{C}$ of the UAC according to the policy. The UAC uses the adapted thresholds $\bar{C}$ to invoke the URSs with the prescribed frequencies.

The selected policy defines the adaptation logic of the URC. At runtime, the URC uses the policy to adapt the frequencies with which the URSs are invoked by the UAC.

Example. In the robotic example, a policy prescribes for each estimated location the thresholds with which the localisation is invoked. According to the selected policy, the URC monitors the robot's estimated location and adapts the threshold. The UAC uses the threshold and invokes the localisation at the prescribes frequency.

We make the following observation for the selected policy: The URC adapts $\bar{C}$ 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 Figure 4(b)). However, in locations that the robot only steps into due to deviations in its movement, such as (3, 2) (shaded orange cell in Figure 4(b)), 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. Figure 4(a)).

4.3 Automated Tool

We instantiate the PARLEY+ methodology and provide an automated tool[7] 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 UAC 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 a module to the PRISM file depicting the URC, as shown in Listing 2. Our tool then generates an initial population of policies with static uncertainty reduction. Afterwards, our tool automatically calls EvoChecker which 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 (RQs)** that we investigate on the robotic use case outlined previously.

[6] https://github.com/carwehlm/PARLEY/tree/v2.0/plots/fronts, and https://github.com/carwehlm/PARLEY/tree/v2.0/Applications/EvoChecker-master/data.
[7] https://github.com/carwehlm/PARLEY/tree/v2.0/.

RQ1. *Effectiveness*: How effective is Parley+'s adaptive uncertainty reduction in terms of achieving the system's objectives (success rate and costs) compared to Parley and a baseline that resolves uncertainty periodically?
RQ2. *Diversity*: How diverse are the policies provided by Parley+ to enable trading off multiple objectives compared to Parley and 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 RQ:

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

Experimental Setup for RQ1 and RQ2. To evaluate the effectiveness and diversity of Parley+ on the robotic use case (Section 3) according to the PRISM model discussed in Section 4, we randomly generate 90 maps of size 10×10. The robot should traverse from the bottom-left to the upper-right corner of the map. Obstacles are placed randomly on the map[8]. Based on the 90 individual maps and their movement controllers, we synthesise 90 PRISM models, as discussed in Section 4. In this context, uncertainty occurs due to probabilistic 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. Section 3). We use the objectives defined in Section 3 and formalised in Section 4.2.3, accordingly.

We pairwise compare Parley+, Parley and 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 and so on[9]. The baseline policies form a Pareto-front concerning the objectives, as depicted by the red crosses in Figure 4(a). In contrast, Parley+ and Parley generate policies that adapt the frequency with which the localisation service is invoked depending on the robot's estimated location. Policies found both by Parley+ and Parley also form a Pareto-front, as depicted by the circles and pluses, respectively, in Figure 4(a).

To quantify the effectiveness (RQ1), we compute the *Hypervolume* [32] that measures the size of the objective space covered by the Pareto-front obtained by each approach. The higher the Hypervolume, more of the objective space is covered meaning that a Pareto-front satisfies better the objectives. Hypervolume is considered a comprehensive quality indicator of a solution set, that is applicable if the number of objectives is rather low, which is true in our case. It is typically used to compare solution sets while a higher Hypervolume indicates a 'better' set in terms of Pareto dominance [32]. We conclude that a larger Hypervolume implies more effective policies to better satisfy the objectives. In our experiments, the Hypervolume computes the area that is covered by a Pareto-front. To this end, we need to set a *reference point* which depicts the worst policy that could potentially be found. We select the worst policy such that it satisfies each objective the least, i.e., if an objective is to be maximised (minimised), the worst policy provides the minimum (maximum) possible value. For instance, in Figure 4(a) we select a policy that has a probability of mission success of 70% while accumulating a cost of 80 as reference point. The Hypervolume of the baseline then represents the area[10] spanned by this point and the red points within the white area. The Hypervolume of Parley+ is represented as the area between the reference point and blue

[8]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.
[9]As discussed in the previous section, we set the maximum interval between two invocations of the service at 10 steps.
[10]Since we have only two objectives, the Hypervolume is computed in the 2D objective space and therefore corresponds to an area.

Table 1. Results for Hypervolume of Baseline Compared to PARLEY (BvP), Baseline Compared to PARLEY+ (BvP+) and PARLEY Compared to PARLEY+ (PvP+) in All Settings, Denoted by Three Values: The Number of Maps in Which the Former Is Significantly Better/Insignificant/Significantly Worse Compared to the Latter

min_success		**max_cost**			
		60	**80**	**100**	**∞**
0%	BvP	05/11/74	01/08/81	01/07/82	00/04/86
	BvP+	00/00/90	00/00/90	00/00/90	00/00/90
	PvP+	00/00/90	00/00/90	00/00/90	00/00/90
60%	BvP	00/00/90	01/00/89	01/00/89	01/00/89
	BvP+	00/00/90	00/00/90	00/00/90	00/00/90
	PvP+	00/02/88	00/02/88	00/00/90	00/00/90
70%	BvP	00/00/90	00/00/90	00/00/90	01/00/89
	BvP+	00/00/90	00/00/90	00/00/90	00/00/90
	PvP+	06/37/47	00/18/72	00/03/87	00/01/89
80%	BvP	00/04/86	00/03/87	00/03/87	02/02/86
	BvP+	00/04/86	00/01/89	00/00/90	00/00/90
	PvP+	10/58/22	00/30/60	00/04/86	00/01/89

points within the white area. In the following, we refer to any policy in terms of the guarantees it provides as (x, y), i.e., $(0.92, 79)$ is a policy that provides a 92% probability to reach the destination and will not incur a cost greater than 79.

Note, that the results reported in this article (cf. Table 1) comparing PARLEY to the baseline do not reproduce the results reported in our previous work [12]. This is due to a miscalculation of the Hypervolume in [12], which positioned PARLEY worse compared to the baseline than PARLEY actually is. Our calculation of the Hypervolume assumes that all objectives are to be minimised. Since we want to maximise the probability p of a mission success, we aim to minimise $1 - p$. However, we did not account for this in the reference point (x_{ref}, y_{ref}) that needs to be adjusted to $(1 - x_{ref}, y_{ref})$. We fixed this mistake in our implementation and present the correct results in this article.

To quantify the diversity (RQ2), we want to analyse the diversity of policies in a Pareto-front. The literature provides the *Spread* [32] metric 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 [32]. Spread is defined as:

$$Spread(A) = \frac{d_{upper} + d_{bottom} + \sum_{i}^{n-1} \left| d_i - \bar{d} \right|}{d_{upper} + d_{bottom} + (n-1) * \bar{d}}, \tag{6}$$

where d_{upper} and d_{bottom} depict the distance between two extreme solutions in the Pareto-front and theoretical endpoints of the solutions. Here we compute these theoretical endpoints as $P_1 = (1.0, y_{ref})$ and $P_2 = (x_{ref}, 0)$.

Spread puts focus mainly on the uniformity of solutions, which is not the main objective of RQ2. For instance, a Pareto-front with only three solutions that are equidistantly spaced along the front might have a very low Spread, whereas a front with 30 solutions that has three clusters, each in the area where the former front only has one point, would have a high Spread since the

solutions are not spaced as uniformly. Still, the latter front might be better to tradeoff objectives for the stakeholders, in particular, because it has more unique solutions. Therefore, we introduce the **Policy Diversity Index (PDI)** which is less susceptible to outliers or an uneven distribution, but still promotes Pareto-fronts that have solutions across the entire Pareto-front:

$$PDI(A) = \frac{d_{upper} + d_{bottom} + \sum_{i}^{n-1} \left| d_i - \bar{d} \right|}{d_{upper} + d_{bottom} + n - 1}.$$

By removing the factor $\bar{d}$, which is the average distance between unique solutions, from the denominator of the Spread metric (cf. Equation (6)), we emphasise more the number n of unique solutions than the evenness of their distribution. We calculate PDI using a reference point (x_{ref}, y_{ref}) to depict the worst policy for a Pareto-front.

For the *a-posterior* analysis of the Pareto-fronts obtained by Parley+, Parley and the baseline, we compute the Hypervolume and PDI concerning practically relevant solutions, which are constrained by individual requirements on the two objectives, as well as the entire Pareto-front. Particularly, we denote with $min\_success \in \{0.0, 0.6, 0.7, 0.8\}$ the minimal success rate (Objective O1 from Section 3) and with $max\_costs \in \{\infty, 100, 80, 60\}$ maximal costs that are required (Objective O2 from Section 3). We consider all combinations of these two requirements, resulting in 16 *settings* to cover a wide range of practically relevant solutions. We depict one such setting with dotted lines in Figure 4(a) and only investigate policies that are within the area of acceptable policies (non-shaded area). When computing the Hypervolume and PDI, we use the boundaries defined by the setting as reference point, i.e., the worst possible solution. When investigating $max\_costs = \infty$, we use the maximum cost of any policy found by any of the approaches as max_costs[11].

To address the stochastic nature of the meta-heuristic search (with EvoChecker[12]) in Parley+ and Parley, we run both Parley+ and Parley 10 times on each map. In contrast, the baseline is deterministic so that one run is sufficient. We further pairwise quantify the gains concerning Hypervolume and PDI from the solutions produced by Parley+, Parley and the baseline and test for statistical significance of the gain using Mann-Whitney U and a 95% confidence level ($p < 0.05$) (cf. [3]). Additionally, we compute Cliff's Delta δ which measures the probability that a randomly selected value from one group is larger than a randomly selected value from the other group, minus the reverse probability. Unlike traditional p-values, Cliff's Delta provides an interpretable measure of how much one distribution dominates another. Values range from -1 to 1, where 0 indicates no difference[13].

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[14], and finally how large the search space is for Parley+. We synthesise 10 maps of each size to account for random outliers and to depict the diversity of maps for each size, and report on the mean and standard deviation, accordingly.

Experimental Setup for RQ4. We investigate RQ4 in two dimensions: We deploy Parley+ to *ROS Gazebo*[15] to demonstrate its feasibility in a realistic setting. We further apply Parley+ to the previously mentioned TAS exemplar [45] to demonstrate its applicability to a different type of system and domain.

[11] We found this to always be the highest cost of the first policy of the baseline, which invokes the URS after every move.
[12] We use EvoChecker out of the box and set a population size of 100 across 40 generations.
[13] The effect size δ of two approaches A and B is usually interpreted as *very small* ($\delta = \pm 0.02$), *medium* ($\delta = \pm 0.52$) or *large* ($\delta = \pm 0.86$) [5], where a positive (negative) value depicts that A (B) outperforms B (A).
[14] We use PRISM in version 4.7 on an M1 MacBook Pro with 16 GB RAM.
[15] Gazebo is a simulator for systems based on the **Robotic Operating System (ROS)**.

5.1 RQ1: Effectiveness

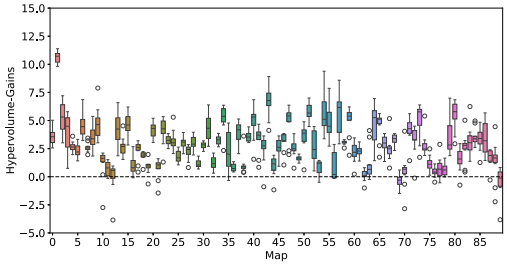
For the RQ1, we investigate if the policies provided by PARLEY+ are better, that is, closer to the optimum (minimal cost and maximum probability of a mission success), than those provided by PARLEY and the baseline. Table 1 reports the results in terms of Hypervolume gains achieved by PARLEY compared to the baseline in the upper entries of each row (BvP). The second row depicts Hypervolume gains from PARLEY+ compared to the baseline (BvP+), while finally, the third row reports a comparison between PARLEY and PARLEY+ (PvP+). For instance, for a minimal success rate of 70% and maximal cost of 80, both PARLEY+ and PARLEY significantly outperform the baseline in 90 out of 90 maps, while PARLEY+ outperforms PARLEY in 72 out of 90 maps. In the remaining 18 maps, there is no statistically significant difference in the Hypervolume of PARLEY+ and PARLEY. Notably, we can observe that across all 1,440 data points, the baseline never outperforms PARLEY+, presumably because we use the baseline as the starting population in the evolutionary algorithm. PARLEY, on the other hand, is outperformed by the baseline in 13 cases in total. Additionally, we can observe that PARLEY improves its performance over PARLEY+ 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 PARLEY+ in 10 out of 90 maps. In contrast, for weak requirements that investigate the entire Pareto-front with $min\_success \geq 0\%$ and $max\_cost \leq \infty$, PARLEY+ provides Pareto-fronts with a significantly larger Hypervolume than PARLEY on all maps. This shows that PARLEY provides valuable solutions in clusters closer to the theoretical optimal policy, whereas PARLEY+ provides a rich Pareto-front over the entire set of possible policies, including extreme points.

On average across all 16 requirement settings (each with 90 maps), PARLEY+ provides statistically significant improvements in 1,435 out of 1,440 cases compared to the baseline and 1,268 out of 1,440 cases compared to PARLEY. Figure 5(a) visualises box-plots of the results for each map for one setting ($min\_success \geq 0\%$ and $max\_cost \leq \infty$) of PARLEY compared to the baseline, while Figure 5(b) shows PARLEY+ compared to the baseline, and Figure 5(c) visualises PARLEY+ against PARLEY[16]. The box sizes indicate that in this setting, PARLEY+ provides more consistent solutions than PARLEY.

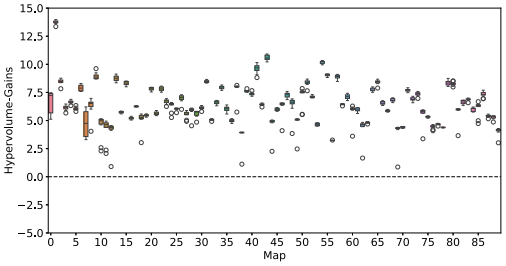
Additionally, Table 2 presents *Cliff's Delta*, computing the effect size of Hypervolume improvement across all 90 scenarios of PARLEY+ (BvP+) and PARLEY (BvP) over the baseline, as well as of PARLEY+ over PARLEY (PvP+). For instance, if we investigate the entire Pareto-front (i.e., the minimal success rate is 0% and the maximal cost is ∞) PARLEY+ has a medium effect size compared to the baseline (0.599), whereas PARLEY only has a small effect size compared to the baseline (0.283). PARLEY+ also has a small effect size over PARLEY (0.365). Investigating only a small portion of the Pareto-front, e.g., with a minimal success rate of 80% and a maximal cost of 60, the effect sizes achieved by both PARLEY+ and PARLEY are much greater (0.633 and 0.635, respectively). In the same case, the effect size between PARLEY+ over PARLEY is positive but negligible (0.004). In all cases, PARLEY+ achieves a positive effect size of the Hypervolume over PARLEY, while both always improve over the baseline.

Compared to the reported results in [12], PARLEY actually performs better as shown in Table 1, outperforming the baseline in 1,389 out of 1,440 (96.5%) cases (or 798 out of the 810 (97.4%) cases considered in [12]), in contrast to the 50% reported in [12].

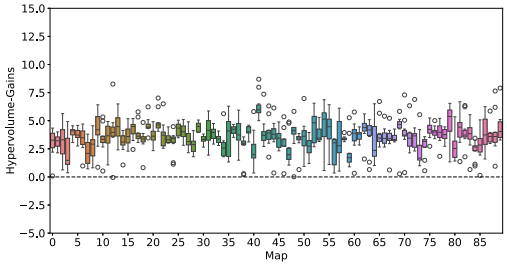
[16]We provide plots for the remaining settings in our replication package https://github.com/carwehlm/PARLEY/tree/v2.0/plots/box-plots.



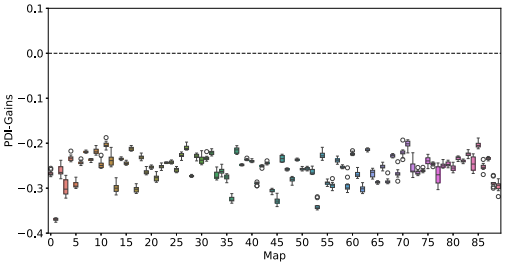
(a) Gain in Hypervolume of Parley compared to Baseline



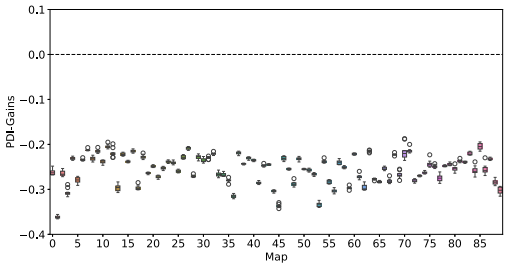
(b) Gain in Hypervolume of Parley+ compared to Baseline



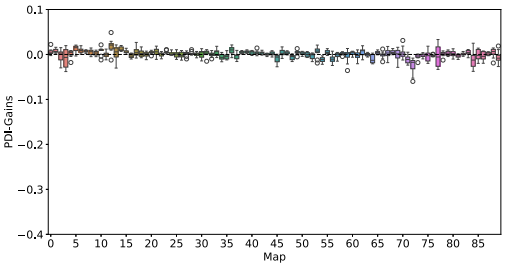
(c) Gain in Hypervolume of Parley+ compared to Parley



(d) Gain in PDI of Parley compared to Baseline



(e) Gain in PDI of Parley+ compared to Baseline



(f) Gain in PDI of Parley+ compared to Parley

Fig. 5. Gain in Hypervolume (higher is better) and PDI (lower is better) across different maps for the entire front, i.e., for setting $min\_success = 0\%$, $max\_cost = \infty$.

We conclude that Parley+ significantly helps to achieve the system's objectives by determining when to reduce uncertainty in 99.7% (1,435/1,440) of the cases compared to the baseline. Additionally, Parley+ outperformed Parley in 88.1% (1,268/1,440) of the cases.

5.2 RQ2: Diversity

For the RQ2, we investigate if the policies generated by Parley+ provide more diverse tradeoffs than those generated by Parley and the baseline. If so, more different tradeoffs along the Pareto-front rather than similar tradeoffs 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 PDI (a lower value indicates a higher diversity). Table 3 reports these results comparing the baseline to Parley (BvP), the baseline to Parley+ (BvP+) and Parley to Parley+ (PvP+) for the 16 requirement settings. For instance, for a minimal success rate of 70% and maximal cost of 80, both Parley and

Table 2. Results for Cliff's Delta in Hypervolume Comparing Parley over the Baseline (BvP), Parley+ over the Baseline (BvP+) and Parley+ over Parley (PvP+) in All Settings

min_success		max_cost			
		60	80	100	∞
0%	BvP	0.334	0.363	0.359	0.283
	BvP+	0.720	0.722	0.711	0.599
	PvP+	0.563	0.523	0.497	0.365
60%	BvP	0.625	0.567	0.503	0.463
	BvP+	0.682	0.638	0.616	0.595
	PvP+	0.123	0.129	0.165	0.182
70%	BvP	0.636	0.570	0.500	0.464
	BvP+	0.645	0.608	0.592	0.581
	PvP+	0.032	0.071	0.124	0.153
80%	BvP	0.633	0.528	0.448	0.409
	BvP+	0.635	0.586	0.570	0.564
	PvP+	0.004	0.063	0.126	0.161

Table 3. Results for PDI of Baseline Compared to Parley (BvP), Baseline Compared to Parley+ (BvP+) and Parley Compared to Parley+ (PvP+) and in All Settings, Denoted by Three Values: the Number of Maps in Which the Former Is Significantly Better/Insignificant/Significantly Worse Compared to the Latter

min_success		max_cost			
		60	80	100	∞
0%	BvP	00/00/90	00/00/90	00/00/90	00/00/90
	BvP+	00/00/90	00/00/90	00/00/90	00/00/90
	PvP+	15/62/13	30/51/09	30/51/09	00/00/90
60%	BvP	00/00/90	00/00/90	00/00/90	00/00/90
	BvP+	00/00/90	00/00/90	00/00/90	00/00/90
	PvP+	01/32/57	00/28/62	00/24/66	00/22/68
70%	BvP	00/00/90	00/00/90	00/00/90	00/00/90
	BvP+	00/00/90	00/00/90	00/00/90	00/00/90
	PvP+	19/57/14	07/61/22	04/56/30	04/46/40
80%	BvP	00/04/86	00/03/87	00/03/87	00/03/87
	BvP+	00/04/86	00/01/89	00/01/89	00/00/90
	PvP+	30/54/06	10/52/28	04/46/40	03/38/49

Parley+ significantly outperform the baseline in all 90 maps. Additionally, Parley outperforms Parley+ in seven maps, whereas Parley+ outperforms Parley in 22 maps. In the remaining 61 maps, there are no statistically significant differences.

Table 4. Results for Cliff's Delta in PDI Comparing Parley over the Baseline (BvP), Parley+ over the Baseline (BvP+) and Parley+ over Parley (PvP+) in All Settings

min_success		max_cost			
		60	80	100	∞
0%	BvP	1.000	1.000	1.000	1.000
	BvP+	1.000	1.000	1.000	1.000
	PvP+	−0.041	−0.100	−0.115	−0.090
60%	BvP	1.000	0.999	0.999	0.999
	BvP+	1.000	1.000	1.000	1.000
	PvP+	0.348	0.380	0.406	0.434
70%	BvP	1.000	1.000	1.000	1.000
	BvP+	1.000	1.000	1.000	1.000
	PvP+	0.019	0.112	0.180	0.241
80%	BvP	0.920	0.891	0.892	0.842
	BvP+	0.922	0.963	0.964	0.946
	PvP+	−0.075	0.046	0.125	0.152

For all settings, Parley+ achieves significantly better results than the baseline in 1,434 out of 1,440 cases. Compared to Parley, Parley+ provides better results in PDI in 526 cases and worse results in 188 cases. Figure 5(d) visualises the gain in PDI (lower is better) achieved by Parley over the baseline for one setting ($min\_success \geq 0\%$ and $max\_cost \leq \infty$), with Figure 5(e) comparing the baseline to Parley+, and finally Figure 5(f) comparing Parley to Parley+.

Additionally, Table 4 presents Cliff's Delta in PDI across all 90 scenarios of Parley+ (BvP+) and Parley (BvP) compared to the baseline, which indicates how much they dominate the baseline. For instance, if we investigate the case of a minimal success rate of 70% and maximal cost of 80, both Parley+ and Parley achieve a large effect size of PDI (1) over the baseline. Parley+ has a small effect size of PDI over Parley (0.112). In general, the effect sizes of PDI achieved by Parley+ and Parley are large (always over 0.842) over the baseline. In five cases, Parley has a better effect size compared to Parley+ (negative value in Table 4). In the other 11 cases, Parley+ outperforms Parley. We observe that the improvements achieved by Parley+ and Parley over the baseline are smaller for a strong requirement on the minimal success rate (>80%) than for weaker requirements as displayed in the table.

While the PDI can be influenced strongly by the number of unique solutions, the results from RQ1 suggest that in most cases Parley+ still covers more of the objective space than the baseline and Parley. Thus, we conclude that the results indicate improved diversity of solutions using Parley+.

We conclude that the policies provided by Parley+ are significantly more diverse than the baseline policies in 99.6% (1,434/1,440) of the cases, which offers a more diverse set of tradeoffs, from which stakeholders can select for self-adaptation. Compared to Parley, Parley+ was able to find significantly more diverse solutions in 526 cases (36.5%), whereas Parley outperformed Parley+ in 188 cases (13.1%).

Table 5. Scalability of PARLEY+

Map Size	#States		Model Checking Time (s)		Search Space
	μ	σ	μ	σ	
5×5	571.8	357.4	0.006	0.004	$5^{25} \approx 10^{17}$
10×10	1,137.2	62.5	0.029	0.015	10^{100}
15×15	2,484.4	111.6	0.094	0.024	$15^{225} \approx 10^{264}$
20×20	4,436.0	186.4	0.311	0.054	$20^{400} \approx 10^{520}$

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. For the number of States and Model Checking times, we generate 10 maps for each map size and report mean (μ) and standard deviation (σ).

5.3 RQ3: Scalability

In the RQ3, 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 policies.

We investigate different map sizes of the running example. Table 5 shows the results. We generate 10 maps of each size to account for outliers, i.e., maps that are not representative of maps of this size. Such outliers might be maps where large areas are blocked by obstacles, reducing the number of reachable states for the system.

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. Considering the evolutionary search to generate policies, for a larger map (20×20) 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 ≈26 minutes ($2{,}500 \times 0.311\,\text{s} \times 2$ properties)[17]. In contrast, model checking every policy with PRISM would take $\approx 10^{520}$ minutes ($20^{400} \times 0.311\,\text{s} \times 2$ properties). Thus, PARLEY+ improves the scalability relying on evolutionary search, which, however, can still be an issue for applying PARLEY+ at runtime in highly dynamic systems.

The scalability might also have implications on the extent of the capabilities to be prescribed to the URC. In our running example, the time for URC synthesis is directly linked with the map size. That means the stakeholders need to determine if the time to synthesise a URC is acceptable. If not, the design of the URC, and thus accompanying model, can be reevaluated to reduce the number of states or the search space. For instance, the abstraction within the discretisation can be less granular resulting in fewer grid cells to be modelled. While this will improve synthesis time, URC performance will likely be lower. Alternatively, several URC s could be synthesised on sub-maps enabling parallel synthesis. At runtime, the URC could be switched depending on where the robot is estimated to be. This, however, poses the threat for locally optimal controllers that miss a global optimal performance w.r.t. the objectives.

[17]Note that in practice, PARLEY+ will need more time caused by additional overhead, e.g., from the evolutionary algorithm.

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 RQ4: Practicality

To investigate the RQ4, we first apply Parley+ in a realistic robotic setting, which expands our running example. Afterwards, we investigate how Parley+ can be applied to a self-adaptive service-based system.

Continuous Robot. Expanding from the problem investigated so far we consider a more realistic setup. Specifically, a Turtlebot3 Waffle[18] must travel to a destination while not crashing into obstacles or leaving the area, using a faulty left wheel. This fault was introduced to have an additional significant challenge for the URC to address beyond the realistic setup. The system now contains a robot operating in continuous space 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 $21\,\text{m} \times 21\,\text{m}$ discretised into $3\,\text{m} \times 3\,\text{m}$ cells, resulting in a 7×7 environment. Therefore, the state space is $x \in [0, 6]$, $y \in [0, 6]$ and $\theta \in \{\text{North, East, South and West}\}$.

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:

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

As in the previous discrete task, the robot can perform one of four actions: move North, South, West or East. To perform an action, the robot will first rotate in place to have the correct heading if not so already, and then drive forward to reach the environment. Therefore, the state new cell. The robot receives movement commands via ROS' network, specifically a Twist message, that includes the linear and rotational velocities. The intended linear and rotational velocities from receiving a command shall be denoted respectively as v and ω.

Given the estimate of the robot's current and desired heading, the difference can be calculated as d_θ. With ω known, the time to complete rotation can be calculated as $t_\theta = \frac{d_\theta}{\omega}$. Once t_θ time has passed the robot receives a velocity command where $\omega = 0$, thus halting the robot's movement. For moving forward, similarly the system is aware of the distance between the centre of the two cells, d_s, and with velocity v the time required to complete the journey is $t_s = \frac{d_s}{v}$.

To include the malfunction of the left wheel, the linear velocities of the right and left wheels, v_r and v_l are required to be calculated. The right wheel's linear velocity can be found using $v_r = v + \frac{w\omega}{2}$ where w is the distance between the wheels. The rotating command will have $v = 0$ and for the forward command $\omega = 0$; therefore:

$$v_r, v_l = \begin{cases} v_r = \frac{w\omega}{2} \\ v_l = -v_r \end{cases} \quad \text{if rotating}$$

$$\qquad\qquad\quad \begin{cases} v_r = v \\ v_l = v_r \end{cases} \quad \text{if forward}$$

As earlier stated, we introduce a fault in the left wheel, which will be the wheel operating at lower power. This will incur significant additional drift, where the aim of the substantially increased difficulty is to gauge the capabilities of Parley+ through comparing to Parley and the baseline.

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

Table 6. Values Used for Velocity Control of the Turtlebot

Parameter	Value
v	0.2 m/s
ω	0.1 rads/s
f	0.99
w	0.287 m

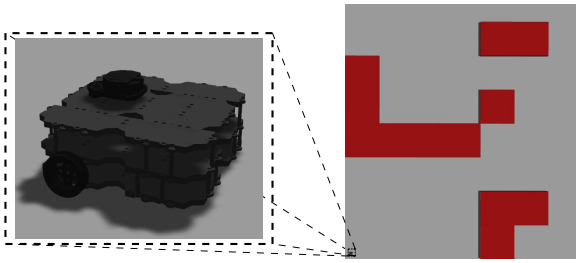


Fig. 6. Simulated setup of the Turtlebot tasked with reaching a goal position. Cell [0, 0] is bottom left, and cell [6, 6] is top right.

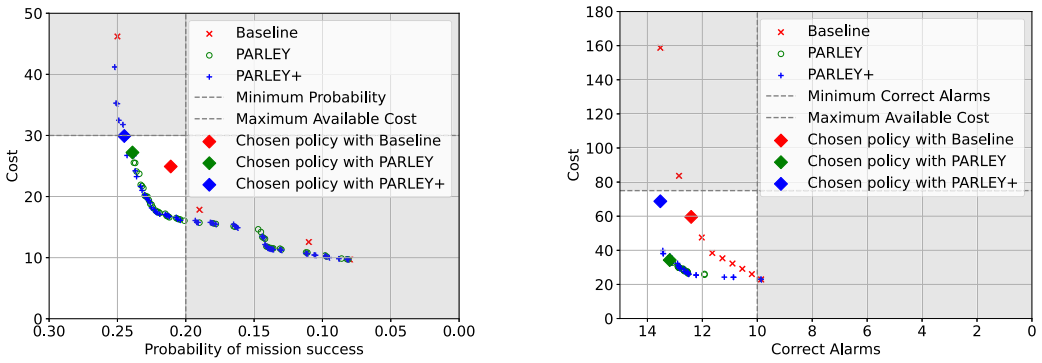
For introducing the fault $\tilde{v}_l = \xi v_l$ where ξ is the effect of the wheel's fault.

Using the faulty value, $\tilde{v}_l$, the actual linear and rotational velocities are calculated as $\tilde{v} = \frac{v_r + \tilde{v}_l}{2}$ and $\tilde{\omega} = \frac{2v_r - 2v}{w} = \frac{v_r - \tilde{v}_l}{w}$, respectively.

The robot is tasked with travelling from cell [1, 0] to cell [3, 5], and 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, where if invoked the robot will be able to align its estimated position with its actual position. The system has the same objectives and incurs the same costs for moving and calling the service as the running example. Here, though, the minimum acceptable probability of succeeding and the maximum allowable total cost are set to 0.2 and 30, respectively. The values of the robot's velocity variables can be found in Table 6.

We conducted simulations in Gazebo using ROS packages, as visualised in Figure 6. To acquire the probability transitions the robot was first initialised in a random cell with a random initial heading, and chose random actions (North, East, South and West) until the robot crashed or left the environment. These traces were recorded to attain the state transitions and the corresponding probabilities. The initialisation of the robot's position would be in the cell's centre, with the heading either 0, $\frac{\pi}{2}$, π and $-\frac{\pi}{2}$, i.e., precisely East/North/West/South. Due to the robot's consistent drift, however, while the robot travelled during this data collection process, we would acquire the transitions when not in the centre as well. This would ensure implicitly of capturing the robot's average position and heading in a cell at any given time, and thus accurately capture the average outcome of the robot's actions. This process was repeated for 10^5 traces.

Once the controllers have been synthesised, these are now parameters for the robot's URC. Specifically here, a text file that is read upon mission start, demonstrating the ease of integrating



(a) Continuous Robot.

(b) Tele-Assistant Service.

Fig. 7. Pareto-fronts for PARLEY+, PARLEY and a baseline in different applications, depicting constraints in the objectives and one chosen policy for each approach.

the synthesised policy into the Turtlebot's controller. The code for running the experiments can be found online, along with a video illustrating the simulation setup[19].

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 Figure 7(a). The results demonstrate the difficulty of the task due to the high drift, however, PARLEY+ provides policies that dominate every policy generated by the baseline. Evidencing clearly that the inclusion of a URC greatly improves the performance of both the overall mission success and cost to execute.

To conclude this particular case study, the results, while gathered from a relatively simple robot setup, do provide insight into the possibilities of PARLEY+ for more complex systems. Robotic systems can have access to a wide variety of sensors to achieve their objectives, such as solving the **Simultaneous Localisation and Mapping (SLAM)** problem. There are a variety of SLAM algorithms and approaches, each with their strengths and weakness'. **Active SLAM (ASLAM)**, for example, is a subset of algorithms which employs search strategies that directs the robots movements for acquiring informative data to solve SLAM. This active element greatly assists for enabling autonomy, however the computational cost of enacting ASLAM is expensive. This approach can have implications on the overall mission as well. The mission, for example, could be time sensitive and the robot performing explorative behaviour during the mission may raise the overall completion time beyond an acceptable limit. Furthermore, there is always the usage of remote operators that can guide the robot if dangerously lost through the use of camera, and can directly input observed landmarks for localisation. These examples highlight the variety of URSs possible, and the costs associated, be it the computational cost of using a particular SLAM algorithm, mission time for using ASLAM or utilising a remote operator's effort. Policies capturing the tradeoffs can be synthesised with PARLEY+, and stakeholders will then have the flexibility of selecting the most appropriate, which here might include minimising remote operator's involvement or minimising overall mission time.

Service-Based System. We use a DTMC model of the TAS system to demonstrate PARLEY+'s ability to synthesise policies for uncertainty reduction for a service-based system. To this end, we model a patient who might be unwell, causing an alarm with a probability of 25% per round. Our TAS model has two equivalent services, one of which should be used to transmit the alarm. Each service can be available or unavailable. The UAC decides, which of the services to use, relying on an estimate

[19] https://github.com/carwehlm/PARLEY/tree/v2.0/turtlebot.mp4.

Table 7. Key Characteristics of the Systems and pDTMC Models Used for Evaluations of PARLEY+

	Discrete Robot	**Continuous Robot**	**Web Application [12]**	**Tele-Assistant Service [45]**
Application domain	Mobile robot navigation		Server infrastructure management	Digital health
System type	CPS	CPS	Multi-server system	Service-based system
pDTMC				
# States	2,977	512	5,630	45,394
# Transitions	4,779	1,070	11,540	104,090
# Parameters	10^{100}	10^{7*7*4}	132	10^{32}

of each service's availability. To transmit an alarm, the UAC invokes the first service if it estimates the service to work correctly, and the second service otherwise. The UAC estimates each service's availability by probing it from time to time. This probing resembles the available URSs. In the baseline, this probing is done with a fixed frequency. We define a reward structure representing the cost for invoking and probing services (cost of 1 for service 1, cost of 2 for service 2) and rewards for correctly invoking a service if an alarm needs to be transmitted (reward of 10 in any case as long as the invoked service is available).

With PARLEY, we set dynamic frequencies for the two services, with our URC taking into account the estimated availabilities as well as whether one of the services was invoked in the previous round (and which one). PARLEY+ additionally uses the baseline with a fixed frequency as part of the starting population for its evolutionary search.

Figure 7(b) displays Pareto-fronts for the baseline as well as PARLEY and PARLEY+. With two services at its disposal that can be probed, we model two *URSs* and thus demonstrate PARLEY+'s ability to find suitable policies for systems with more than one *URS*.

Summary. Table 7 displays the main characteristics of all systems we investigated in this work and previously in [12]. While the robots are CPS, the web application that we investigated in [12] 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. Additionally, the service-based TAS system shows that PARLEY+ can also handle even larger models than shown previously in [12].

We have successfully applied PARLEY+ to two additional systems: a realistic use case of a real continuous robot (Turtlebot3), and a service-based application from literature. PARLEY+ was able to find a substantial amount of unique policies that dominate the baseline in both applications. This provides evidence about the practicality of PARLEY+ in different domains and complexities 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 10 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. We construct 10 maps for each size to account for that threat. The same threat 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., [24, 30] for more information on their scalability. Currently, Parley uses PRISM, which limits its scalability. Using EvoChecker [19], we started addressing this concern but still limited the map size. Additionally, we modelled the URSs 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 four systems from three application domains (cf. Table 7) to mitigate the threat of generalisability. The diversity in application domain across these four systems provides evidence about the versatile use of Parley+ and indicates that an application of Parley+ to additional application domains can lead to similar positive results as presented in Table 7.

Conclusion. We used the Mann-Whitney U test pairwise for significant differences between Parley+, Parley and the baseline with a 95% confidence level and relied on the extensive guidelines in [3] 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 16 different settings covering all combinations of weaker and stronger requirements for minimal success rate and maximal cost as well as the entire Pareto-front.

6 Conclusion and Future Work

In this work, we motivated the need for a controller dedicated to reducing epistemic uncertainty. We extended Parley, an end-to-end methodology and architecture that relies on a pDTMC system model which synthesises an URC. This clearly separates 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.

To this end, we presented Parley+. This work also extends the formalisation, capturing the URC synthesis. We prove that our model augmentation, implementing this synthesis, produces a valid model that can simulate the original model. Similar to Parley, our extension Parley+ 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. Here, Parley+ improves the existing approach by utilising an improved initial population for generating Pareto-optimal controllers.

Parley+ was evaluated on 90 instances of the robotic example with 16 different requirement settings, and compared to Parley and a baseline concerning effectiveness and diversity. The results show that Parley+ provides significantly improved controllers in 88.1% of all 1,440 cases in terms of effectiveness compared to Parley, and 99.7% compared to the baseline, with the baseline never significantly outperforming Parley+. For the diversity, Parley+ presented significant improvements over Parley in 36.5% of all 1,440 cases, whereas in 13.1% of all 1,440 cases Parley significantly outperformed Parley+. Thus, Parley+ enables a richer and more diverse set of tradeoffs, from which the stakeholders can choose for self-adaptation. Compared to the baseline, both Parley+ and Parley provide significant improvements in the majority of cases, as a result from the increased number of unique solutions that both approaches generate compared to the baseline. Finally, we demonstrated Parley+'s practicality by applying it to four different systems from three different domains, including a service-based system 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 four different systems, in future work we aim to apply PARLEY+ to more systems of other domains. To this end, we plan to model the controller's estimations with probabilistic distributions, which is already possible within our extended formalisation. 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 URSs, 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/tree/v2.0/.

Acknowledgements

The authors acknowledge help in the implementation by Arturo Bertoglia and the application of PARLEY on TAS by Maximilian Orthmann.

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 *Proceedings of the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 3389–3396. DOI: https://doi.org/10.1109/IROS.2014.6943034

[2] Naif Alasmari, Radu Calinescu, Colin Paterson, and Raffaela Mirandola. 2022. Quantitative verification with adaptive uncertainty reduction. *Journal of Systems and Software* 188 (2022), 111275. DOI: https://doi.org/10.1016/j.jss.2022.111275

[3] 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. DOI: https://doi.org/10.1002/stvr.1486

[4] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking*. MIT Press.

[5] Mattan S. Ben-Shachar, Daniel Lüdecke, and Dominique Makowski. 2020. effectsize: Estimation of effect size indices and standardized parameters. *Journal of Open Source Software* 5, 56 (2020), 2815.

[6] Juergen Branke, Kalyan Deb, Henning Dierolf, and Matthias Osswald. 2004. Finding Knees in Multi-Objective Optimization. In *Parallel Problem Solving from Nature - PPSN VIII*. PPSN 2004. Lecture Notes in Computer Science, Vol. 3242, Springer, Berlin. DOI: https://doi.org/10.1007/978-3-540-30217-9_73

[7] Yuriy Brun, Ron Desmarais, Kurt Geihs, Marin Litoiu, Antonia Lopes, Mary Shaw, and Michael Smit. 2013. *A Design Space for Self-Adaptive Systems*. Springer, Berlin, 33–50. DOI: https://doi.org/10.1007/978-3-642-35813-5_2

[8] Ricardo Diniz Caldas, Arthur Rodrigues, Eric Bernd Gil, Genaína Nunes Rodrigues, Thomas Vogel, and Patrizio Pelliccione. 2020. A hybrid approach combining control theory and AI for engineering self-adaptive systems. In *Proceedings of the 2020 IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 9–19. DOI: https://doi.org/10.1145/3387939.3391595

[9] 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. DOI: https://doi.org/10.1109/TSE.2010.92

[10] Radu Calinescu, Raffaela Mirandola, Diego Perez-Palacin, and Danny Weyns. 2020. Understanding uncertainty in self-adaptive systems. In *Proceedings of the IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS '20)*. IEEE, 242–251. DOI: https://doi.org/10.1109/ACSOS49614.2020.00047

[11] Matteo Camilli, Raffaela Mirandola, and Patrizia Scandurra. 2022. Taming model uncertainty in self-adaptive systems using Bayesian model averaging. In *Proceedings of the 2022 IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 25–35. DOI: https://doi.org/10.1145/3524844.3528056

[12] Marc Carwehl, Calum Imrie, Thomas Vogel, Genaína Rodrigues, Radu Calinescu, and Lars Grunske. 2024. Formal synthesis of uncertainty reduction controllers. In *Proceedings of the 19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '24)*. ACM, New York, NY, 2–13. DOI: https://doi.org/10.1145/3643915.3644095

[13] 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. DOI: https://doi.org/10.1016/j.scico.2018.07.002

[14] 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.

[15] Thomas Vogel (Ed.). 2024. Software Engineering for Self-Adaptive Systems Exemplars Repository. Retrieved from http://self-adaptive.org/exemplars/

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

[17] Luis Garcia, Huma Samin, and Nelly Bencomo. 2024. Decision making for self-adaptation based on partially observable satisfaction of non-functional requirements. *ACM Transactions on Autonomous and Adaptive Systems* 19, 2, Article 11 (Apr. 2024), 44 pages. DOI: https://doi.org/10.1145/3643889

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

[19] 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.

[20] Carlo Ghezzi, Mauro Pezzè, Michele Sama, and Giordano Tamburrelli. 2014. Mining behavior models from user-intensive web applications. In *Proceedings of the 36th International Conference on Software Engineering (ICSE '14)*. ACM, New York, NY, 277–287. DOI: https://doi.org/10.1145/2568225.2568234

[21] Ruben Giaquinta, Ruth Hoffmann, Murray Ireland, Alice Miller, and Gethin Norman. 2018. Strategy synthesis for autonomous agents using PRISM. In *NASA Formal Methods*. AaronDutle, CésarMuñoz, and AnthonyNarkawicz (Eds.), Springer International Publishing, Cham, 220–236.

[22] 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*. Nelly Bencomo, Robert France, Betty H. C. Cheng, and Uwe Aßmann (Eds.), Lecture Notes in Computer Science, Vol. 8378, Springer, 47–100. DOI: https://doi.org/10.1007/978-3-319-08915-7_3

[23] Edwin González, Javier Sanchis, José Vicente Salcedo, and Miguel Andrés Martínez. 2023. Conditional scenario-based model predictive control. *Journal of the Franklin Institute* 360, 10 (2023), 6880–6905. DOI: https://doi.org/10.1016/j.jfranklin.2023.05.012

[24] 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.

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

[26] 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.

[27] Jeff Kramer and Jeff Magee. 1990. The evolving philosophers problem: Dynamic change management. *IEEE Transactions on Software Engineering* 16, 11 (1990), 1293–1306.

[28] Jeff Kramer and Jeff Magee. 2007. Self-managed systems: An architectural challenge. In *Future of Software Engineering (FOSE '07)*. IEEE, 259–268. DOI: https://doi.org/10.1109/FOSE.2007.19

[29] Andreas Kreutz, Gereon Weiss, and Mario Trapp. 2022. Towards uncertainty reduction tactics for behavior adaptation. In *Proceedings of the European Conference on Software Architecture*. Springer, 199–214.

[30] Marta Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of probabilistic real-time systems. In *Proceedings of the23rd International Conference on Computer Aided Verification (CAV '11)*. Springer, 585–591.

[31] 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. DOI: https://doi.org/10.1109/TRO.2022.3200138

[32] 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. DOI: https://doi.org/10.1109/TSE.2020.3036108

[33] Kasper S. Luckow and Corina S. Pasareanu. 2016. Log2model: Inferring behavioral models from log data. In *Proceedings of the 2016 IEEE International High Level Design Validation and Test Workshop (HLDVT)*, 25–29. DOI: https://doi.org/10.1109/HLDVT.2016.7748251

[34] 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 Transactions on Autonomous and Adaptive Systems* 15, 4 (2020), 10:1–10:36. DOI: https://doi.org/10.1145/3487921

[35] 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.

[36] 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 *Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 7344–7350.

[37] 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 (SEAMS '18)*. ACM, New York, NY, 51–57. DOI: https://doi.org/10.1145/3194133.3194144

[38] 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.

[39] Yi Qin, Yanxiang Tong, Yifei Xu, Chun Cao, and Xiaoxing Ma. 2024. Active monitoring mechanism for control-based self-adaptive systems. *Proceedings of the ACM on Software Engineering* 1, FSE, Article 82 (July 2024), 24 pages. DOI: https://doi.org/10.1145/3660789

[40] Andres J. Ramirez, Adam C. Jensen, and Betty H. C. Cheng. 2012. A taxonomy of uncertainty for dynamically adaptive systems. In *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '12)*. HausiA. Müller and LucianoBaresi (Eds.), IEEE Computer Society, 99–108. DOI: https://doi.org/10.1109/SEAMS.2012.6224396

[41] Huma Samin, Nelly Bencomo, and Peter Sawyer. 2022. Decision-making under uncertainty: Be aware of your priorities. *Software and Systems Modeling* 21, 6 (Dec. 2022), 2213–2242. DOI: https://doi.org/10.1007/s10270-021-00956-0

[42] Raquel Sánchez Salas, Javier Troya, and Javier Cámara. 2025. Automated planning for task-based cyber-physical systems under multiple sources of uncertainty. In *ACM Transactions on Autonomous Adaptive Systems*. DOI: https://doi.org/10.1145/3733603

[43] Stepan Shevtsov, Danny Weyns, and Martina Maggio. 2019. SimCA*: A control-theoretic approach to handle uncertainty in self-adaptive systems with guarantees. *ACM Transactions on Autonomous and Adaptive Systems* 13, 4, Article 17 (July 2019), 34 pages. DOI: https://doi.org/10.1145/3328730

[44] 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 *Proceedings of the 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 89–99.

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

[46] Danny Weyns, Radu Calinescu, Raffaela Mirandola, Kenji Tei, Maribel Acosta, Amel Bennaceur, Nicolas Boltz, Tomás Bures, Javier Cámara, et al. 2023. Towards a research agenda for understanding and managing uncertainty in self-adaptive systems. *ACM SIGSOFT Software Engineering Notes* 48, 4 (2023), 20–36. DOI: https://doi.org/10.1145/3617946.3617951

[47] 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.

[48] 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.

[49] Danny Weyns and Usman M. Iftikhar. 2023. ActivFORMS: A formally founded model-based approach to engineer self-adaptive systems. *ACM Transactions on Software Engineering and Methodology* 32, 1, Article 12 (Feb. 2023), 48 pages. DOI: https://doi.org/10.1145/3522585

[50] 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. *Requirements Engineering* 15, 2 (2010), 177–196. DOI: https://doi.org/10.1007/S00766-010-0101-0

Received 18 October 2024; revised 20 April 2025; accepted 20 June 2025