Towards the Specification of Adaptive Robotic Systems

Jeremy Morse, Dejanira Araiza-Illan, Jonathan Lawry and Kerstin Eder †

March 4, 2016

Abstract

The widespread adoption of autonomous adaptive systems depends on provided guarantees of safety and functional correctness, at both design time and runtime. Specifying adaptive systems is cognitively difficult when their aspects are in a large number and have complicated dependencies.

We present a technique to construct and automatically explore a specification for systems that can degrade and/or adapt, towards analysis at design time for verification and validation. This technique combines and constructs sections of a lattice (or Hasse diagram) of all the possible ordered system degradations/adaptations of interest, limited by desirability or risk thresholds. The lattice allows the designer to understand the different levels and combinations of system degradations/adaptations. We use the lattices (or sections) to systematically explore whether a system is able to fulfil its task goals under a dynamic and uncertain environment, through probabilistic model checking.

We illustrate the proposed specification technique through a domestic robotic assistant example. Systematically exploring the lattice allowed comparing the probabilities of task success/failure, to determine which degradation/adaptation combinations can be allowed in the final system implementation.

1 INTRODUCTION

Fully autonomous systems interacting with complex environments require adaptive behaviour, when the system degrades due to usage and malfunctions, when requirements vary, or when the environment is uncertain and dynamic. Ideally,
an autonomous robot meant to operate in the real world for a long period of
time should be able to cope with its own degradation and most of the changes
in its environment. Autonomous robotic systems need enough flexibility to re-
spond to changes in the environment, to achieve its ultimate goal as best as
possible [1].

Traditional specification conventions (e.g., formal such as temporal logics [2,
timed input/output automata [3] or the Z language [4], or more conventional
such as flow charts or requirement documents [5, 6]) target non-adaptive sys-
tems or systems where all the adaptations can be clearly identified, through
exhaustively enumerating all functional possibilities and environmental scenar-
ios in models. These models can be subjected to proof or test to determine if
the system designs are fit for the tasks and goals they need to achieve. Such
techniques are not suitable in dynamic environments, however, as all possible
environmental changes must be anticipated a priori and their effects on the
requirements analysed. This becomes cognitively difficult when aspects of the
model have complicated dependencies: for example an obstacle may require
a robot to reduce speed to maintain a safety margin, affecting it’s ability to
perform it’s function timely. Furthermore, traditional specification frameworks
cannot express flexibility in the requirements themselves. It may be desirable,
for example, to trade a slight relaxation in one safety requirement for a dramatic
improvement in others.

We wish to ask quantitative questions such as “what is the maximum amount
of degradation that a system can accept and still successfully complete a task
within a reasonable threshold of resources”, identifying the weakest possible
operational point of the robot while still meeting requirements. Being able
to automatically analyse and predict a priori how a degrading and/or adaptive
system would cope when its circumstances are altered would open opportunities
for the development of new verification and validation techniques at design
time. We ask whether it is possible to build specifications that express such
degradation or adaptation, that can be used as the basis for automatic design
space exploration.

For example, suppose a domestic robotic assistant such as the Care-O-Bot[1]
has capabilities that degrade due to constant use, such as battery capacity, or
increased joint resistance from dust accumulation. Additionally, the robot must
adapt to the preferences of the user, who might be elderly or have chronic health
issues [7]. If the robot must frequently remind a person to take vital medicine
doses, but also periodically recharge batteries, a design decision must be made of
the maximum velocity the robot can operate at and the minimum battery level
before recharging. These choices must be balanced against the risk of collision,
of flattening it’s batteries, and of missing a scheduled medicine reminder. The
acceptable risk levels of such occurrences is a matter for validation, and may
prompt exploration of the design space.

Most of the research on adaptive systems has focused on developing runtime
methods. By monitoring the satisfaction of requirements, adaptations are cho-

---
[1]http://www.care-o-bot-4.de/
sen and triggered. Without verifying these adaptive systems before deployment, it is assumed that the triggered adaptations will be suitable in their context, or that for each non-satisfied requirement, there will be suitable adaptation. A remaining challenge is to be able to verify adaptive systems at design (or requirements) time, to forecast what the adaptations and degradations will do (e.g., worst case scenarios). Different formal models have been proposed to try to capture the adaptations of software systems, to be able to verify the code at design time. Nonetheless, describing what the system can do is still mostly hand-crafted through exhaustive enumeration, which is prone to errors.

In this paper, we focus on the problem of specifying systems that can degrade and/or adapt, that can be explored in an automated way that allows in principle enumerating all the valid combinations of the system’s possible degradations and/or adaptations. We present a technique to construct and automatically explore a specification, from ordered sets of degradations/adaptations for individual variables, system parameters or behaviours (e.g., changes of constraints over specific parameters or functionalities). This technique combines and constructs sections of a lattice (or Hasse diagram) of all the possible system degradations/adaptations of interest, with each point representing desirability or risk values. The lattice allows the designer to understand the different levels and combinations of system degradations/adaptations and their impact, enabling analysis at design time for verification and validation. We use lattices constructed through the proposed technique to systematically explore these ordered degradations/adaptations during system design, to discover whether a system is able to fulfil its task goals in combination with models of a dynamic and uncertain environment. In particular, we illustrate the use of the technique for exploration of a domestic robotic assistant’s specification.

Determining the desirability or risk value of a lattice node is performed by composing a model of the dynamic environment with the operational point of the lattice node. We use PRISM, a probabilistic model checker\(^2\) to explore the environmental model, combining a nondeterministic robot controller with a probabilistic environment. When presented with an operational point, the model checker finds the worst possible outcome for the best possible robot behaviour, i.e., assuming a robot that always makes decisions with the maximum probability of success, what is the worst outcome, while operating at the given point?

This technique produces a design-space lattice with associated desirability / risk values. Selecting a particular point on the lattice is then a matter for the designer, deciding whether the given degradation/adaption point is acceptable to achieve the associated desirability/risk value.

Systematically exploring the lattice allows comparing the probabilities of task success/failure, to determine which degradation and/or adaptation combinations are suitable or will be allowed in the final system implementation. Our algorithm and the exemplified system design exploration approach can be extended to other types of systems beyond robotic assistants, and even runtime

\(^2\)http://www.prismmodelchecker.org/
verification, in principle.

The paper proceeds as follows. Section 2 introduces the algorithm to automatically construct ordered specifications for degrading or adaptive systems. Section 3 presents a case study of a domestic robotic assistant, and its verification at design time through model checking, based on preferences over the ordered system specification. In Section 4, we verified some properties, which provided insights on preferred adaptations and worst case scenarios. We discuss the presented approach in Section 5. Section 6 presents an overview of related work on the specification and verification of adaptive systems. We conclude the paper in Section 7 and give an outlook on future work.

2 SPECIFICATION OF SYSTEMS THAT DEGRADE

2.1 A Lattice to Capture Degrading System Behaviour

We present the computation of a lattice to specify a degrading/adaptive system, based on a simple starting specification. This avoids having to enumerate and order these sets of possible degradations/adaptations by hand.

Let \( L \) be a language of propositional logic with a finite set of propositional variables \( P \), and connectives, \( \wedge \), \( \lor \) and \( \neg \). Let \( SL \) denote the sentences of \( L \). A specification is formed by propositional logic expressions, such as “if \( x \geq 0.25 \), then \( y < 0.5 \)”, i.e.

\[
\psi = \neg p_2 \lor \neg q_3.
\] (1)

Definition Let \( S \) be a finite set of valuations on \( SL \) representing the set of possible states of the world (the adaptive system and its environment), or our current knowledge about that set.

Definition Entailment means that for \( \theta, \phi \in SL \), \( \theta \models_S \phi \) (\( \theta \) entails \( \phi \)) if and only if \( \forall v \in S, v(\theta) = 1 \Rightarrow v(\phi) = 1 \).

Definition The set of degradations/adaptations of a sentence are defined as \( \mathcal{W}(\psi) = \{ p_i \in P : p_i \} \), for \( p_i \in P \), that for propositional variables \( \mathcal{W}(p_i) \), it can be totally ordered according to \( \models_S \).

Definition A partial ordering \( \theta \leq \phi \) exists if and only if \( \theta \in \mathcal{W}(\phi) \).

Definition Let \( \Psi \) be a specification as in (1), then the degraded/adapted specification is the poset \( (\mathcal{W}(\psi), \leq) \).

We build a lattice or Hasse diagram for the poset, for exploration purposes. The general goal of the construction and exploration is to identify the maximal elements of \( (\mathcal{W}(\psi) \cap \{ \theta : \models_S \theta \}, \leq) \), which can be interpreted as finding “the best adaptation under degradation”, or “the worst degradation allowed”. We could employ other criteria to choose between competing maximal elements (i.e.,
those at the same lattice level), such as a cost function associated to risk, or the shortest sequences of degradations/adaptations in the lattice.

To exemplify the lattice construction, consider a system with two variables \( x \) and \( y \), both taking values in \([0, 1]\). We then define the following propositional variables: \( p_1 = x \geq 0 \), \( p_2 = x \geq 0.25 \), \( p_3 = x \geq 0.5 \), \( p_4 = x \geq 0.75 \), \( p_5 = x = 1 \), and \( q_1 = y \geq 0 \), \( q_2 = y \geq 0.25 \), \( q_3 = y \geq 0.5 \), \( q_4 = y \geq 0.75 \), \( q_5 = y = 1 \). This would be equivalent to enumerating all the variables, and their individual degradations/adaptations. The degradations/adaptations for each proposition are: \( \mathbb{W}(p_1) = p_1 \), \( \mathbb{W}(p_2) = p_1, p_2 \), \( \mathbb{W}(p_3) = p_1, p_2, p_3 \), \( \mathbb{W}(p_4) = p_1, p_2, p_3, p_4 \), \( \mathbb{W}(p_5) = p_1, p_2, p_3, p_4, p_5 \), and \( \mathbb{W}(q_1) = q_1 \), \( \mathbb{W}(q_2) = q_1, q_2 \), \( \mathbb{W}(q_3) = q_1, q_2, q_3 \), \( \mathbb{W}(q_4) = q_1, q_2, q_3, q_4 \), \( \mathbb{W}(q_5) = q_1, q_2, q_3, q_4, q_5 \). Also, \( \mathbb{W}(\neg p_2) = \neg p_2, \neg p_3, \neg p_4, \neg p_5 \), and \( \mathbb{W}(\neg q_3) = \neg q_3, \neg q_4, \neg q_5 \). Thus, \( \mathbb{W}(\neg p_2 \lor \neg q_3) = \{ \neg p_2 \lor \neg q_3, \neg p_2 \lor \neg q_4, \neg p_2 \lor \neg q_5, \neg p_3 \lor \neg q_3, \neg p_3 \lor \neg q_4, \neg p_4 \lor \neg q_3, \neg p_4 \lor \neg q_4, \neg p_4 \lor \neg q_5, \neg p_5 \lor \neg q_3, \neg p_5 \lor \neg q_4, \neg p_5 \lor \neg q_5 \} \), for the specification \( \psi = \neg p_2 \lor \neg q_3 \). Instead of enumerating all these possible combinations, which becomes intractable for larger problems, a lattice is computed. As shown in Fig. 1, Acceptable (closest to the upper) degradations/adaptations to explore, would be, e.g., \( \neg p_3 \lor \neg q_3 \) and \( \neg p_2 \lor \neg q_4 \).

The pseudo-algorithm for the Hasse diagram (or lattice) computation is shown in Fig. 2. We compute the lattice breadth first, one degradation/adaptation level at a time, by enumerating all possible successive degradation/adaptation combinations. Consider a clause \( C = \bigvee_{i=1}^{k} l_i \) where \( l_i \neq l_j \) and \( l_i \neq \neg l_j \), for \( i \neq j \). \( k \) is the number of degradations/adaptations per parameter, variable or behaviour. Let \( \mathbb{W}(l_i) = \{ l_{i,1}, \ldots, l_{i,k_i} \} \), where \( l_{i,k_i} = l_i \) and \( l_{i,s} \leq l_{i,r} \), \( s \leq r \). Then, if \( C' \in \mathbb{W}(C) \), \( C' = \bigvee_{j=1}^{k} j_i \), where \( j_i \in \{1, \ldots, k_i\} \) for \( i = 1, \ldots, k \). Let the degradation/adaptation degree of the literal \( l_i \) in clause \( C' \) be \( D(l_i, C') = k_{i,j_i} \). Let the degradation/adaptation level of \( C' \) be \( DL(C') = \sum_{i=1}^{k} D(l_i, C') \). In the Hasse diagram there will be a link between \( C'' \) and \( C' \) if and only if \( DL(C'') = DL(C') + 1 \) and \( \max\{ D(l_i, C'') D(l_i, C') : i \} = 1 \).

Avoiding the whole lattice exploration (i.e. not computing the whole lattice from the start) obeys the need to keep degradations/adaptations in realistic and desirable levels (e.g. avoiding instances where the safest adaptation is not doing
1: Specification $\psi$, ordered propositions e.g. $p_m, q_n$, with $m, n$ degradation/adaptation degrees, respectively
2: for $k = \max\{m, n\}$ do
3: Enumerate lattice nodes (e.g. $C'$) at the same degradation/adaptation level, e.g., $p_m$ and $q_n$ s.t. $DL = m + n$
4: Build links between nodes at different levels, e.g. $C''$ and $C'$
5: end for

Figure 2: Pseudo-algorithm to compute the lattice

anything at all, which is undesirable for a reactive robot). Consequently, constraints can be introduced when building the lattice, to limit the computed links between levels according to the relations between degradations/adaptations and desirability/reality thresholds.

The complexity of the algorithm to compute the Hasse diagram is dependent on the size of the different degradation/adaptation levels. The number of elements in level $n$ is bounded above by the quantity of natural number solutions to the equation

$$\sum_{i=1}^{k} x_i = n. \quad (2)$$

This upper bound is achieved when $|W(l_i)| \geq n$ for $i = 1, \ldots, k$.

2.2 Model

Determining the desirability / risk value for a particular lattice node requires evaluating a model of the robot in an environment, while operating at the given operational point. To achieve this we construct models for use in the PRISM probabilistic model checker, which supports verification of properties given in various temporal logics and quantifying the probability of a particular property holding. The model is formed of three parts: the environment, the robot, and a property generated for each lattice node.

The environmental model is created as a discrete time Markov chain written in the PRISM modelling language, with variables for all relevant parts of the environment. Non-robot actors are then specified using probabilistic behaviour to represent uncertainty about their actions. For example, a human in a 2D space may be specified as moving stochastically, with no particular purpose. All dynamic or uncertain behaviours must be encoded in the environmental model, to allow opportunity for the model checker to explore different outcomes based on those behaviours.

The robot controller is also written in the PRISM modelling language, but is not defined to have a particular implementation. Rather than specifying how the robot works in all circumstances, we instead specify the set of actions that it may choose to take, as nondeterministic choices. The model checker is then able to pick which action the robot is to take, in any particular circumstance.
In essence, we give the model checker an abstract robot controller, and it then synthesises a controller appropriate to the environmental model. The combined environmental and robot models form a Markov decision process, or MDP.

Finally, we produce a verification property for each lattice node, representing an operational point. This is written as a PRISM “filter” statement. We pre-suppose a formula for the success of the robot activity using the “Pmax” operator of PCTL, a probabilistic temporal logic. We also pre-suppose a set of start states, representing the range of normal operational states of the robot, and use the “min” filter operator to select the state with the lowest probability of the formula succeeding. The constraints represented by the operational point are added to the start state preconditions and to the property formula as a global invariant, limiting the set of choices available to the robot while completing the activity.

Combining the environmental and robot models with the verification property causes PRISM to search for the set of choices for the robot behaviour that maximises the chance of the formula being true. The “min” operator then picks the lowest of these probabilities. The result is then, for the given configuration, the worst-case probability of success of the robot performing at that operational point.

3 CASE STUDY: MODEL CHECKING TO VERIFY PROPERTIES OF A DOMESTIC ROBOTIC ASSISTANT UNDER A SPECIFIC DEGRADATION

We considered a domestic robotic assistant in an open-plan, confined floor of a house, represented by a grid. The robot is allowed to move in four directions, north, south, east or west in the grid, with limited maximum velocity, or it can choose to stay in the same cell. A human cohabits this space, also allowed to move stochastically in the grid. A representation of this setup is shown in Fig. 3.

The robot has a battery, and it will seek recharge at a station fixed in the grid, when reaching a minimal energy threshold. The battery energy diminishes each time the robot moves (one unit per motion cycle). The robot needs to coordinate recharging, whilst servicing the person within a time threshold. Additionally, the robot avoids colliding with the person for safety.

For the case study, we would like to determine how quickly the robot can service the human with a particular maximum velocity, and what margin of energy reserves are required to complete this task. Firstly, we computed a lattice with the system parameters that can be adapted or degraded over its operation and lifetime: the maximum velocity for the robot, \( p_i = v \leq i \) with \( i = 1, \ldots, 6 \), and the maximum allowed time to service the human, \( q_j = j \leq t \) with \( j = 1, \ldots, 10 \).

\[
\psi = p_i \land q_i
\]  

(3)
In an ideal world, the robot would be able to service the human as fast as possible, and moving at a velocity that is not dangerous, at $v \leq 1 \land t \leq 1$ units. Nevertheless, if this is not possible, we could settle for a suboptimal option, e.g. $v \leq 3 \land t \leq 5$ units. We ask what are the best suboptimal options, and what is the worst case?

To answer these questions, an enumeration of options is necessary, followed by analysis mechanisms, e.g., computing the probability of achieving the human servicing task given degraded/adapted thresholds. The lattice of the specification was expanded completely, since we only used a small set of variables in the case study. Fig. 4 shows only a segment of the resulting lattice, from the ideal specification to degradations/adaptations at level 5. A full expansion might not be convenient for larger problems and state spaces of degradations/adaptations, and sections of it can be computed instead. Secondly, from each node in the lattice to be explored – i.e. a particular set of valuations (instantiations) for the degraded/adapted parameters –, we model checked a parametrized model of the robot-human system in PRISM.

In classical model checking, a finite-state model of a system is explored exhaustively, to determine if a temporal logic property is satisfied or not. PRISM targets systems that exhibit non-deterministic behaviour, modelled as discrete-time Markov chains, continuous-time Markov chains, or Markov decision processes (MDPs), among others. Probabilities of property satisfaction are com-
The main drawback of model checking is the cost in computational time and memory, due to the state space explosion problem. This problem is due to the exponential growth in number of states to traverse, when adding variables [8]. So far, the computational demands of model checking have made it unsuitable for runtime verification [9].

We constructed a parametrized model of the domestic robot setting described before (with a person as the stochastic environment), in the form of Markov decision processes. This model contains 5 modules, corresponding to the human and robot motion, the timing, the energy in the battery, and the servicing task. The parametrization prunes the state space of the model to the particular degradation/adaptation, which otherwise would comprise all possible limits for the parameters. Also, specifying initial conditions, such as the initial battery charge, reduces the state space. The model checker computes a minimum probability to complete the servicing of the human, subjected to particular values of initial battery energy, maximum velocity for the robot, and maximum allowed time to service the human. When the robot does not need to service the person, the model checker chooses a velocity and direction for its motion, from the available options, including the specified velocity upper limit. Also, the robot can reach the battery recharge station either moving from north to south, or from east to west, choosing randomly. The human moves stochastically at all times in the grid.

The temporal logic property specification language for MDPs is probabilistic computation tree logic (PCTL). PRISM allows qualitative questions about the probability ($P$) of properties (e.g., the probability of event $X > 0.9$, true or false?), and also quantitative questions, i.e., computing actual probabilities (e.g., what is the maximum probability of event $X$, given some initial conditions?). Allowed PCTL operators (to describe events happening along execution paths in the model) include: $\land$ (next, or within the next time step), $\lor$ (until, or an event is true until another event is true), $\lnot$ (eventually, or sometime in the future), $\neg$ (always, or along all execution paths), $\forall$ (weak until, which, besides the $\lor$ operation, allows the first event to always be true if the second is never true) and $\exists$ (release, where an event is true until another true, or the first event is true forever). Bounded time can be added to the operators, e.g.
$a \quad \mathcal{U}_{\leq t} \quad b$ is satisfied if $b$ becomes true within $t$ steps, and $a$ is true in all the states before $t$.

The degradations are parametrized into the model for model checking, through the “filter” construct in PRISM attached to the properties to verify. The filters choose, from all the possible states, the ones where the specified parametrization is true. This helps reducing the state space explosion, pruning the exploration and saving time and computational resources. Additional filters have been added to eliminate other invalid or unrealistic model behaviours, such as when the system does not perform any action at all over time, which lead to false conclusions. This is equivalent to constrain the system to avoid the cases where a controller chooses to not do anything ever, to comply with “always being safe” or “never doing something wrong”.

An example of a property to verify, or a query about the model under a particular degradation/adaptation is: “computing the minimum probability of completing a servicing of the human in 5 robot motion steps, considering a specific instance of servicing time threshold bound, the maximum velocity for the robot, the minimum energy threshold, and the starting energy”.

\[
\begin{align*}
\text{filter}(\min, P_{\max} &=?((\text{serviceHuman}) \land (velocity \leq 4) \\
\mathcal{U}_{\leq 20}(\neg \text{serviceHuman}) \land (velocity \leq 4)), \\
((\text{serviceHuman} \land \text{serviceTimer} = 0) \lor \neg \text{serviceHuman}) \\
\land (velocity \leq 4) \land (\text{minenergy} = 2) \land (\text{energy} = 25) \\
\land (\text{tick} = 0) \land ((\text{robotX} \neq \text{humanX}) \lor (\text{robotY} \neq \text{humanY})))
\end{align*}
\] (4)

4 EXPERIMENTS AND RESULTS

The lattice was computed through a Python script. We ran the model checking experiments on a PC with Intel i5-3230M 2.60 GHz CPU, 8 GB of RAM, Ubuntu 14.04, and PRISM 4.2.beta1. Model checking took less than 1 minute for each experiment, with a minimal time of 10 seconds.

The graphs in Fig. 5 show the probability of servicing the human, for different degradations/adaptations comprising maximum velocity and maximum servicing time. The grid size is $7 \times 7$ cells. The minimum threshold for battery recharging ($min = 2$ units), the initial locations of the human and the robot in the grid (at $(5, 5)$ and $(4, 4)$, respectively), and the location of the recharge station (at $(0, 0)$ or top left corner), were left at specified fixed values. The initial battery energy charge was varied for each graph, from the set $\{25, 20, 15, 10, 5, 4, 3, 2, 1\}$ units, respectively.

The graphs show that the probabilities of satisfying (4) under the different degradations/adaptations for the maximum velocity and the maximum allowed servicing time are reduced considerably when the initial battery charge drops below 5 units. Also, we can observe that the robotic assistant will be unable to satisfy the property for the ideal condition where the velocity is minimal (i.e. $v \leq 1$) and the servicing time is minimal too (i.e. $t \leq 1$). Nonetheless, we could settle for degradation/adaptation thresholds of $v \leq 4$ and $t \leq 4$ (lattice node
Figure 5: Probabilities when degrading the maximum velocity limit for the robot, and the maximum servicing time cycles, for different initial battery energy charges

\( p_4 \land q_4 \), \( v \leq 3 \) and \( t \leq 5 \) (lattice node \( p_3 \land q_3 \)), or \( v \leq 2 \) and \( t \leq 6 \) (lattice node \( p_2 \land q_6 \)), all at level 6 of degradation/adaptation, if the battery is initially charged at above 5 units. For safety reasons, as the velocity is less, we would prefer \( p_2 \land q_6 \) over the other two options. Another critical threshold would be to start with a battery charged at above 4 units, since degrading/adapting to \( p_4 \land q_4 \) or \( p_3 \land q_5 \) provides an acceptable probability of success.

5 DISCUSSION

Our specification technique has the benefits provided by automation, for the computation of the lattice: avoids the enumeration of all possible degradations/adaptations by hand, reduces the presence of errors due to manual input, and it is applicable to many case studies.

Computing a lattice of specifications presents information about the adaptive system possibilities to the designer. Furthermore, exploring the lattice exposes tradeoffs between choices in degradations/adaptations. An ordered lattice provides expressiveness to specify systems that degrade or adapt in uncertain environments, and meaning to exploration results (e.g., thresholds of what is acceptable).

Our approach can be extended through providing new functions to order the elements in the lattice, to express preferences. Also, although the lattice can be computed totally, providing enough computational resources and motivation, segments of it can be computed and explored instead according to relevance or preference.
Another advantage of our technique is the incorporation of formal analysis methods during the exploration phase. Formal methods can provide proof of requirement satisfaction or violation, which will guarantee the safety and functional correctness of a model of the system and its environment, under degradation/adaptation. The adoption of non-deterministic parametrized models entails that choices over the system’s and environmental actions remain specified as open as possible. The model checker decides what are the best actions that the system can perform, according to the non-deterministic environment, under a particular degradation/adaptation.

At the same time, the use of model checking brings the state-space explosion problem, which impedes the exploration of models with a large number of variables or possible states. Nonetheless, solving the state-space explosion problem is a major contemporary research topic in the formal methods community. Furthermore, we would like to examine the substitution of model checking by approximation and numerical methods based approaches.

6 RELATED WORK

Research on requirements engineering seeks to formulate requirements that correctly reflect a system’s adaptation. These approaches could be extended to deal with adaptive system specification. Fuzzy logic, probabilistic, and risk analysis models can be used to formulate the system’s requirements when uncertainty is present due to adaptation [11, 12, 13]. Qualitative requirements, or soft-goals, which mean to be abstract to be applicable to a system that changes, can also be modelled through fuzzy logic [14].

In contrast, [15] argues that the requirements should be formulated as “hard” but contextualized (e.g., tolerances for the requirements according to the environment), instead of fuzzy. This approach would allow verification of adaptive systems through existing formal methods. Nonetheless, all the tolerances and contexts would have to be enumerated and explored at some point, for which no proposal is provided.

Other research on requirements engineering has focused on extracting the requirements that are satisfied by a system. Constraints can be removed systematically, to provide new requirements that can be satisfied by a system, as it is done for linear temporal logic properties in [16].

Transitions between configurations of adaptive software have been represented through Markov chains, Petri Nets, dynamic decision networks, UML diagrams, labelled transition systems, communicating sequential processes and set theory [17, 18, 19]. Nonetheless, these models require enumeration and construction of all possible configuration adaptations, particularly for formal analysis at design stage.

Domain specific languages have been developed to specify adaptive software. In [19], the variability of the software is modelled by specifying, in first-order logic, variation points, which alternatives are available on each point, and constraints to indicate which variants (configurations) are valid. Properties to be
optimized during adaptation are also defined, with priority rules to trigger particular property optimization according to contexts. Our approach does not require any domain specific language, and its scope is more general than software. Also, system configurations are ranked according to property satisfaction in [19], whereas we order the different degradations or adaptations according to their desirability levels. Additionally, our approach allows to add other cost functions over the adaptations/degradations such as risk.

Adaptive systems have been verified at design time or at runtime [18] [13]. At design time, a fixed set of operators for adaptation [20] have been verified through model checking. Model checking has also been employed to verify software adaptive systems modelled as labelled transition systems, communicating sequential processes, set theory or probabilistic timed automata [18]. Models and implementations of adaptive systems have been tested in simulation [18] [21], including stochastic ones [22].

At runtime, adaptive systems are monitored to see if they satisfy requirements, or to trigger new adaptations otherwise. Monitors for high-level or abstract requirements, which are “generalized” to be applicable to any adaptation by the system, have been proposed in [23]. Also, monitors have been used to estimate probabilities of property satisfaction, via updating a Hidden Markov Model with new observed information in [24]. In [25], monitors based on process algebras are proposed, for requirements that need expressing concurrency. In [26], a model of a distributed software adaptive system and its environment is constructed in Z, which is modified according to feedback from monitoring the verification of requirements at runtime (reflecting about what has happened, and changing accordingly). Consequently, this will trigger adaptations to the real distributed system.

Formal methods at runtime, particularly model checking, have been employed to trigger adaptation to satisfy a requirement. If-then-else rules that change have been verified [9] in the model checker SPIN [3]. Probabilistic parametrized models for controllers that change have been verified in PRISM [9]. Both of these model checking at runtime approaches are computationally expensive, thus not feasible for realistic systems.

Another application of formal methods is synthesis of controllers or strategies, or refinement. For example, the best strategies or models of a system that satisfy a property are computed in [27]. Strategies that violate a property the least are computed in [28], from weighted automata and reward assignment. Nonetheless, synthesis and refinement processes at runtime are also computationally expensive [9]. These synthesis processes do not necessarily entail adaptation (as observed in [29]), as the systems to control normally are not allowed to modify their operational spaces, nor is the environment allowed to change dynamically, or non-deterministically.

By analysing a system at design time, it is possible to predict or give some guidance with respect to what are the best or worst case scenarios. Furthermore, we can determine what might happen when a system encounters rare, extreme,
and interesting circumstances. Most of the existing approaches to deal with uncertainty and change in the environment are designed for runtime \cite{19}; i.e., these systems will wait until something happens to adapt in response, and it might be too late to prevent undesired adaptations or degradations.

In principle, our approach could be used at runtime, although today this is limited by the use of formal methods (in particular model checking), which are computationally expensive. In the future this limitation may be overcome by dedicated and efficient computational approaches instead of using model checkers.

7 CONCLUSIONS

We presented a technique to construct and automatically explore a specification for systems that can degrade and/or adapt. This technique combines and constructs sections of a lattice (or Hasse diagram) of all the possible system degradations/adaptations of interest, as needed, limited by desirability or risk thresholds. The system degradations/adaptations can refer to individual variables, system parameters, or behaviours (e.g., changes of constraints over specific parameters). The lattice allows the designer to understand the different levels and combinations of system degradations/adaptations, following their specified order/preference, towards applying these to analysis at design time for verification and validation.

In this paper, we used the lattices (or lattice sections) to systematically explore ordered degradations/adaptations during system design (i.e., before deployment or runtime), to find out if a system is able to fulfil its task goals in combination with models of a dynamic and uncertain environment. We illustrated this exploration through a domestic robotic assistant case study. For the case study, we implemented a parametrized non-deterministic model of a robotic assistant and its home environment (including people), based on Markov decision processes (MDP) in PRISM. This model encodes all possible robot degradations and adaptations (i.e., all the possible “controllers” and “worlds”), such as losing battery life, or changing its maximum speed limits. Providing a degree of degradation/adaptation of the system from the lattice (i.e., a lattice node), parametrization takes place in the model. The (minimum) probability of the degraded/adapted system to satisfy a task goal (expressed as a temporal logic property), given the dynamic and uncertain environment, and specified initial conditions, is computed automatically by the model checker. Systematically exploring the lattice allowed comparing the probabilities of task success/failure, to determine which degradation and/or adaptation combinations were suitable or allowed in the final system implementation.

Whereas for small case studies, the whole lattice can be computed and explored and even visualized, partial segments of interest of a full lattice can be used instead for the analysis of real-life complex systems. In the near future, we will be applying this premise to a real autonomous system, for exploration and verification at design time. Additionally, we will research the application of our
technique to runtime verification of adaptive systems.

References

[1] G. Sussman, “Building robust systems an essay,” in MIT, 2007.

[2] P. Bellini, R. Mattolini, and P. Nesi, “Temporal llogic for real-time system specification,” ACM Computing Surveys, vol. 32, no. 1, pp. 12–42, 2000.

[3] A. David, K. Larsen, A. Legay, U. Nyman, and A. Wasowski, “Methodologies for specification of real-time systems using timed I/O automata,” in Proc. FMCO, 2010, pp. 290–310.

[4] J. Spivey, “An introduction to Z and formal specifications,” Software Engineering Journal, vol. 4, no. 1, pp. 40–50, 1989.

[5] D. Hatley and I. Pirbhai, Strategies For Real-Time System Specification. Dorset House, 1988.

[6] J. Bowen and M. Hinchey, High-integrity system specification and design. Springer-Verlag, 1999.

[7] K. Kucher and D. Weyns, “A self-adaptive software system to support elderly care,” in Proc. MIT, 2013.

[8] E. Clarke, W. Klieber, M. Nováček, and P. Zuliani, “Model checking and the state explosion problem,” in Proc. LASER, 2011, pp. 1–30.

[9] R. Calinescu and S. Kikuchi, “Formal methods at runtime,” in Proc. Monterey Workshops, 2010, pp. 122–135.

[10] M. Kwiatkowska, G. Norman, and D. Parker, “PRISM 4.0: Verification of probabilistic real-time systems,” in Proc. 23rd International Conference on Computer Aided Verification (CAV’11), ser. LNCS, vol. 6806. Springer, 2011.

[11] P. Sawyer, N. Bencomo, J. Whittle, E. Leiter, and A. Finkelstein, “Requirements-aware systems,” in Proc. RE, 2010, pp. 95–103.

[12] J. Whittle, P. Sawyer, N. Bencomo, B. Cheng, and J. Bruel, “RELAX: a language to address uncertainty in self-adaptive systems requirement,” Requirements Engineering, vol. 15, no. 2, pp. 177–196, 2010.

[13] Z. Yang, Z. Li, Z. Jin, and Y. Chen, “A systematic literature review of requirements modeling and analysis for self-adaptive systems,” in Proc. REFSQ, 2014, pp. 55–71.

[14] M. Serrano, M. Serrano, and J. Sampaio do Prado Leite, “Dealing with softgoals at runtime: A fuzzy logic approach,” in REatRunTime, 2011.
[15] A. Chopra, “Requirements-driven adaptation: Compliance, context, uncertainty, and systems,” in Proc. REatRunTime, 2011, pp. 32–36.

[16] K. Kim, G. Fainekos, and S. Sankaranarayanan, “On the revision problem of specification automata,” in Proc. ICRA, 2012, pp. 5171–5176.

[17] R. Bruni, A. Corradini, F. Gadducci, A. Lluch Lafuente, and A. Vandin, “A conceptual framework for adaptation,” in Proc. FASE, 2012, pp. 240–254.

[18] D. Weyns, “Towards an integrated approach for validating qualities of self-adaptive systems,” in Proc. WODA, 2012, pp. 24–29.

[19] F. Fleurey and A. Solberg, “A domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems,” in Proc. MODELS, 2009, pp. 606–621.

[20] D. Gordon, “Asimovian adaptive agents,” Journal of Artificial Intelligence Research, vol. 13, pp. 95–153, 2000.

[21] J. Câmara, R. de Lemos, N. Laranjeiro, R. Ventura, and M. Vieira, “Robustness evaluation of the Rainbow framework for self-adaptation,” in Proc. SAC, 2014, pp. 376–383.

[22] C. Rouff, R. Buskens, L. Pullum, X. Cui, and M. Hinchey, “The AdaptiV approach to verification of adaptive systems,” in Proc. C3S2E, 2012, pp. 118–122.

[23] B. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, “Software engineering for self-adaptive systems: A research roadmap,” Software Engineering for Self-Adaptive Systems, pp. 1–26, 2009.

[24] E. Bartocci, R. Grosu, A. Karmarkar, S. Smolka, S. Stoller, E. Zadok, and J. Seyster, “Adaptive runtime verification,” in Proc. RV, 2012, pp. 168–182.

[25] B. Abolhasanzadeh and S. Jailili, “Towards modeling and runtime verification of self-organizing systems,” Expert Systems with Applications, vol. 44, pp. 230–244, 2016.

[26] D. Weyns, S. Malek, and J. Andersson, “FORMS: Unifying reference model for formal specification of distributed self-adaptive systems,” ACM Transactions on Autonomous and Adaptive Systems, vol. 7, no. 1, 2012.

[27] W. Damm and B. Finkbeiner, “Does it pay to extend the perimeter of a world model?” in Proc. FM, 2011, pp. 12–26.

[28] J. Tumova, L. Castro, S. Karaman, E. Frazzoli, and D. Rus, “Minimum-violation LTL planning with conflicting specifications,” in Proc. ACC, 2013, pp. 200–205.
[29] H. Kress-Gazit, G. Fainekos, and G. Pappas, “Temporal-logic-based reactive mission and motion planning,” IEEE Transactions on Robotics, vol. 25, no. 6, pp. 1370–1381, 2009.