# Run-Time Adaptation of Quality Attributes for Automated Planning

Rebekka Wohlrab, Rômulo Meira-Góes
wohlrab@cmu.edu,romulo@cmu.edu
Institute for Software Research
Carnegie Mellon University
Pittsburgh, USA

Michael Vierhauser
michael.vierhauser@jku.at
LIT Secure and Correct Systems Lab
Johannes Kepler University Linz
Linz, Austria

## ABSTRACT

Self-adaptive systems typically operate in heterogeneous environments and need to optimize their behavior based on a variety of quality attributes to meet stakeholders' needs. During adaptation planning, these quality attributes are considered in the form of constraints, describing requirements that must be fulfilled, and utility functions, which are used to select an optimal plan among several alternatives. Up until now, most automated planning approaches are not designed to adapt quality attributes, their priorities, and their trade-offs at run time. Instead, both utility functions and constraints are commonly defined at design time. There exists a clear lack of run-time mechanisms that support their adaptation in response to changes in the environment or in stakeholders' preferences. In this paper, we present initial work that combines automated planning and adaptation of quality attributes to address this gap. The approach helps to semi-automatically adjust utility functions and constraints based on changes at run time. We present a preliminary experimental evaluation that indicates that our approach can provide plans with higher utility values while fulfilling changed or added constraints. We conclude this paper with our envisioned research outlook and plans for future empirical studies.

## CCS CONCEPTS

• **Computer systems organization** → *Robotic autonomy*; • **Hardware** → *Safety critical systems*; • **Theory of computation** → **Verification by model checking**.

## KEYWORDS

quality attributes, automated planning, non-functional requirements, self-adaptation, conflict resolution, constraints

**ACM Reference Format:**
Rebekka Wohlrab, Rômulo Meira-Góes and Michael Vierhauser. 2022. Run-Time Adaptation of Quality Attributes for Automated Planning. In *17th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '22), May 18–23, 2022, PITTSBURGH, PA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3524844.3528063

## 1 INTRODUCTION

Real-world, self-adaptive systems commonly operate in heterogeneous run-time environments. Depending on the context and the task of a self-adaptive system, a variety of quality attributes need to be taken into account that are typically specified by diverse stakeholders [46]. For instance, client-server applications need to consider performance, cost, and reliability concerns [44] to fulfill the requirements of end users and business owners. To plan how the behavior or structure of a self-adaptive system should be adapted in response to changes in the environment, automated planning approaches have been introduced [20, 31] that take these quality attributes into consideration. Automated planning has been successfully applied, for instance, to robotic systems [1], machine tool calibration [32], and urban traffic management [29]. Typically, these planning approaches rely on a *utility function*, i.e., a single aggregate function indicating the utility or satisfaction level related to relevant quality attributes (such as timeliness, safety, or energy efficiency). Utility functions are widely used to specify optimization objectives for adaptation planning and have been successfully applied to self-adaptive systems in the past [8, 10, 15, 19, 40, 40]. In case multiple candidate plans for adaptation exist, as is often the case in practice, the one with the highest expected utility value is selected by the automated planner. Besides utility functions, *constraints* are commonly used to define requirements that must not be violated, e.g., that the system must preserve a minimum battery level to prevent it from running out of energy. These constraints are used to generate possible plans, among which the optimal one is chosen based on the expected utility.

When applying automated planning in practice, utility functions and constraints are typically specified before the system's execution. While goal-oriented approaches have been proposed to capture a system's objectives [2, 14, 30, 35], few techniques exist that support the lightweight adaptation of utility functions and constraints depending on the system's context and changing stakeholder needs. Changes in stakeholder preferences and in a system's environment, however, might require a reprioritization and adjustment of quality attributes [27, 37], including the resolution of conflicts between constraints. This need for quality attribute adjustment is not generally fulfilled by self-adaptive systems [22].

In this paper, we present an approach for the adaptation of quality attributes for automated planning at run time. It relies on mechanisms to semi-automatically adapt constraints and utility functions based on changes in a system's environment or stakeholder input. We demonstrate the feasibility of our approach by applying it to a robotic system and present a preliminary experimental evaluation alongside our envisioned research agenda.

The remainder of this paper is structured as follows: Section 2 presents an example scenario and Section 3 introduces concepts for planning. In Section 4, we describe our approach, whose evaluation we present in Section 5. We further describe our envisioned research outlook in Section 6, followed by related work in Section 7.

## 2 MOTIVATING EXAMPLE

To motivate our approach, we use the example of a robot that performs a series of tasks, for example, moving goods as part of a Cyber-Physical Production System.

Figure 1 illustrates a scenario of a robot repeatedly moving from location ① to ⑥ (e.g., to pick up items at a rack at ⑥).

In our mission planning example, we focus on three quality attributes: *safety*, to avoid collisions with obstacles; *privacy*, to not intrude humans' personal spaces; and the *travel time* of the robot when executing a mission. An automated planner is used to generate policies that take these quality attributes into account. The planner relies on a utility function whose value should be maximized by the generated plan. The utility depends on the costs in terms of safety, privacy, and travel time. In the example, the safety cost is 1 for each traversed semi-occluded path segment and 2 for each traversed occluded path segment. The privacy cost is 1 for each semi-private location and 2 for each private location that is visited. The travel time is set to the traveled distance (indicated by the labels of the edges in Figure 1) multiplied by a speed factor (which can be either 0.5 or 1.0 in the example). Based on the cost $c_i(p)$ of a quality attribute $QA_i$, the utility of a policy $p$ in terms of that quality attribute is defined as $u_i(p) = 1 - c_i(p)/max_{o \in \Pi}\{c_i(o)\}$, indicating how "bad" the cost of $p$ is in comparison to the maximum cost for any possible generated policy $o$. The overall utility of $p$ is then calculated using utility function weights $w_i$, which indicate the importance of each quality attribute. The utility is defined as:

$$u(p) = \Sigma_{i \in QA} w_i u_i(p)$$

In our example, the initial utility function weights are set to 0.333 for safety, privacy, and travel time, i.e., all quality attributes have equal importance. As we describe in Section 5, when using equal utility function weights, the optimal path in the example would be via ④ and ⑤. Besides utility function weights, constraints may need to be considered when generating a set of possible plans at run time. In this paper, we consider the following kinds of constraints:

(1) bounding measures constraints (e.g., setting a deadline of 3.5 time units for a task, to meet business owner requirements);
(2) proximity constraints (e.g., avoiding that a robot moves to a location that is too close to a human or another robot);
(3) speed limit constraints (to comply with safety regulations).

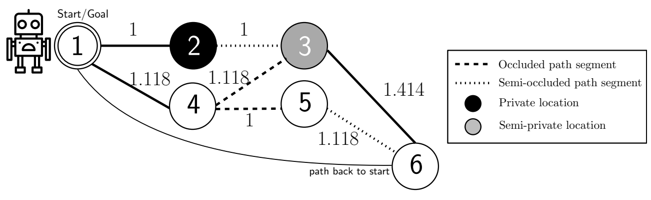

**Figure 1: Example of a robotic mission planning context**

In our example, both utility functions and constraints might need to be adjusted in response to four kinds of possible run-time events:

(1) *Map changes:* a human enters the area and locations need to be avoided for privacy or safety reasons;
(2) *Task changes:* the task is changed to transport a fragile item, which requires new constraints and a higher priority of safety;
(3) *Stakeholder preference changes:* the utility function has to be adjusted to better meet stakeholders' changing preferences;
(4) *Regulatory or restricting changes:* new restrictions or regulations may result in constraints being added, updated, or removed.

For the remainder of this paper, we consider the following scenario: The robot's task changes and it needs to deliver a valuable or dangerous item that increases the importance of safety. To accommodate for this change, the weight of safety is changed to 0.9, the weight of travel time to 0.1, and the weight of privacy to 0. Moreover, a key stakeholder enters the map and makes location ⑤ unavailable (imposing a proximity constraint). Visiting the location should not only be penalized, but must not occur. These changes have an impact on the selected plans, as we describe in our evaluation (Section 5).

## 3 PRELIMINARIES

In this section, we provide a brief introduction to the underlying concepts and the formalism that our approach relies on.

*Markov Decision Processes.* Our automated planning approach builds upon techniques in Markov Decision Processes (MDPs) [24]. An MDP is a tuple $M := (Q, Act, P, q_0, L, R)$, where $Q$ is a finite set of states, $q_0 \in Q$ is an initial state, $Act$ is a finite set of actions, $P : Q \times Act \times Q \rightarrow [0, 1]$ is a probability transition function, $L : Q \rightarrow 2^{AP}$ is a labeling function that maps states to a set of atomic propositions in $AP$, and $R$ is a finite set of cost functions $\rho : Q \times Act \rightarrow \mathbb{R}_{\geq 0}$ that associate non-negative values to every state and action pair.

Intuitively, the labeling function and the cost functions are used to define the quality attributes involved in the planning process. For example, the labeling function is used to define constraints, such as to avoid certain locations in the map. Cost functions quantify the policy and we can define constraints so that the cost is within a certain interval.

*Policies.* A policy for MDP $M$ resolves the nondeterministic choices in $M$ by selecting an action to take in every state. Although there are multiple classes of policies, in this work, we use deterministic memoryless policies. Formally, a policy in $M$ is defined as $p : Q \rightarrow Act$ that maps states into actions. The set of all policies of $M$ is denoted by $\Pi$. Under policy $p$, the behavior of $M$ is fully probabilistic and it can be represented by an induced discrete-time Markov chain [24].

*Temporal properties.* To synthesize policies that satisfy certain objectives and constraints, we utilize the framework of probabilistic temporal logic PCTL, which is used to quantify properties related to probabilities and rewards in system specifications modeled as MDPs [5, 12]. For example, the objective of the robot in Figure 1 is to reach state ⑥ while minimizing travel time, intrusiveness, and collision (maximizing safety), which can be described as a PCTL

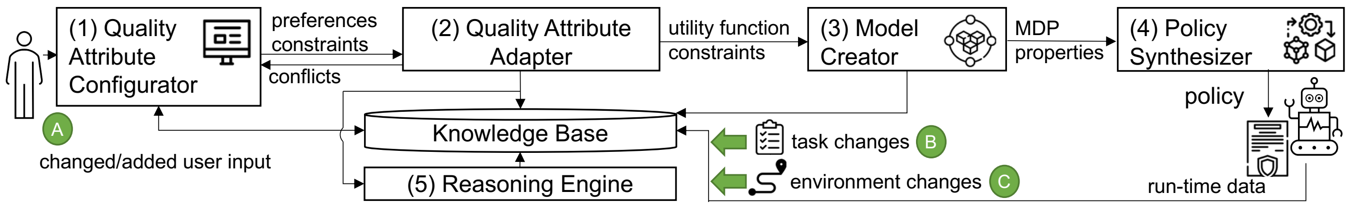

**Figure 2: High-level overview of the main components of our approach**

formula. Formally, the syntax of PCTL for an MDP $M$ is defined as:

$$\phi ::= true \mid c \mid \phi \land \phi \mid \neg\phi \mid \mathcal{P}_{\sim b}[\psi] \mid \mathcal{R}^{\rho}_{\sim r}[C] \mid \mathcal{R}^{\rho}_{\sim r}[F\phi]$$
$$\psi ::= X\phi \mid \phi U^{\leq k}\phi \mid \phi U\phi$$

where $c \in AP$ is an atomic proposition, $\sim \in \{\leq, <, \geq, >\}$, $b \in [0, 1]$ is a probability bound, $r \in \mathbb{R}_{\geq 0}$ is a reward bound, and $\rho \in R$ is a cost function.

The semantics of PCTL formulas are formally defined over the policies of $M$, e.g., see [5, 12]. Herein, we provide intuition on their semantics using a few examples. The operator $\mathcal{P}_{\leq b}[\psi]$ specifies that the probability of taking a path starting in $q_0$ that satisfies property $\psi$ is smaller or equal than $b$ for all policies $p$. Similarly, the reward operator $\mathcal{R}^{\rho}_{\sim r}[C]$ establishes that the expected cumulative cost for cost function $\rho$ is $\sim r$ for all policies. Lastly, the reward operator $\mathcal{R}^{\rho}_{\sim r}[F\phi]$ holds if the total expected cost for cost function $\rho$ before reaching a state that satisfies $\phi$ is $\sim r$ for all policies.

We also allow to replace $\sim$ with $min =?$ or $max =?$ to specify the calculation of the minimum/maximum probability (or reward) over all MDP policies.

## 4 APPROACH

Figure 2 provides an overview of the main parts of our approach for run-time quality attribute adaptation: (1) a *Quality Attribute Configurator* that facilitates user input; (2) a *Quality Attribute Adapter* responsible for analyzing run-time data and user input, resolving conflicts between constraints, and defining the utility function and constraints to be used by the system; (3) a *Model Creator* that takes the provided constraints and utility function as an input and generates an MDP and PCTL properties; (4) a *Policy Synthesizer* that uses the MDP and PCTL properties to generate a mission policy that can then be executed by a self-adaptive system. The Quality Attribute Adapter uses (5) a *Reasoning Engine* to analyze and plan changes to the utility function and constraints, relying on collected run-time data and triggered changes. In our scenario, run-time data refers to any data that can be observed and collected from a running system, such as the current trajectory of a robot, quality measures, sensor data, or detected objects/persons in the environment.

As part of our approach, we currently consider the following changes: (A) new user input, for example, related to changed preferences or constraints, (B) task changes, and (C) changes to the environment (cf. Section 2). A central knowledge base is used to capture user preferences, constraints, utility functions, current tasks, and the current state of the environment in the form of run-time models. This knowledge base is continuously updated based on collected run-time data and user input.

The run-time adaptation of quality attributes can be used to replan at run time, when parts of the plan are being executed. For instance, if any of the listed changes occur when the robot in our running example is at location ④, replanning with the new input can help to decide which path should be selected for the remainder of the mission. Note that run-time replanning can be costly. We discuss ways to address this issue in Section 6.

In the following, we describe the components in further detail.

### 4.1 Quality Attribute Configurator

Before a mission is executed, stakeholders can specify preferences and constraints for a set of quality attributes. Preferences are used to indicate their priorities, whereas constraints can be defined to specify restrictions (such as upper and lower bounds) or invariants that should be guaranteed by the system. For instance, users might want to constrain the battery level to always be above 10% (ensuring that the robot is not running out of energy, so that a mission can be safely completed). The Quality Attribute Configurator also serves as a dashboard, presenting preferences and constraints in a consolidated view, requesting user input in case of conflicts, and providing insights into the system's state and behavior at run time.

### 4.2 Quality Attribute Adapter

The Quality Attribute Adapter executes a reasoning engine to generate a utility function and constraints that should be used during planning. In this paper, we assume the utility function to be a weighted sum of quality attributes [42], where the weights indicate the priorities of each quality attribute. The quality attribute adaptation is based on the indicated preferences and constraints from stakeholders, which are stored in the knowledge base. If safety-related conflicts between constraints occur or it is not possible to create a utility function based on the user input, stakeholders are prompted for input. The call for active human participation is motivated by safety standards requiring human assessment of changes that affect safety-critical constraints. Once all input has been consolidated, the utility function and constraints are used by the Model Creator.

### 4.3 Model Creator

The Model Creator develops the MDP and PCTL properties to be used during policy synthesis. The MDP and properties are created based on the current task that should be executed, a model of the current environment/plan, for example, a map of waypoints, and the previously defined utility functions and constraints.

Using information provided by stakeholders about the environment (Knowledge Base in Figure 2), the Model Creator creates an MDP model which includes (at least) a state for each location in the map, actions to transition between states (e.g., to move to another location or change the speed), and a reward structure capturing the cost obtained in each state for each quality attribute. In other words, the MDP models the environment in which the planning task takes place. In the following, we describe how constraints and costs are captured in the verification property and in reward structures.

*4.3.1 Representation of quality attribute priorities.* We consider weighted sum utility functions in this paper, where the utility function weights $w_i$ indicate the priorities or importance of quality attributes. Approaches for defining weighted sum utility functions for self-adaptive systems based on stakeholder preferences have been previously proposed, e.g., in [46]. When creating the MDP model for policy synthesis, each quality attribute is coupled to a cost function. In this manner, we consider a multi-objective cost function $c$ of MDP $M$ as a linear scalarization of all cost functions: $c(q, a) = \sum_{i \in QA} w_i c_i(q, a)$, where $w_i$ are the utility function weights. To avoid that certain QAs have too much impact on the results, the cost functions $c_i$ are normalized as in [41]. The PCTL formula $\mathcal{R}^{c}_{\min}[C]$ specifies that $c$ should be minimized.

*4.3.2 Representation of constraints.* While the cost function $c$ based on quality attributes ranks the policy planning space, a set of constraints restricts this policy space. In other words, a set of constraints excludes policies that violate any of the specified constraints. For example, constraining the robot to avoid location ⑤ in Figure 1 limits its actions in location ④, i.e., it cannot move to location ⑤. In this work, we consider two types of constraints: safety constraints and bounding cost functions. These constraints support the properties listed in Section 2: bounding measures constraints, proximity constraints, and speed limit constraints.

*Safety constraints:* Safety constraints are related to safety properties in model checking [26]. Note that not all safety properties are safety-critical constraints. Informally, safety properties specify that something "bad" should never occur. Safety properties can introduce proximity constraints and speed limit constraints. For example, constraining the robot to avoid location ⑤ in Figure 1 is a safety constraint. In this case, any visit to location ⑤ is "bad". All policies that avoid this location satisfy this safety constraint.

To formally define safety constraints, we introduce PCTL safety formulas that the robot's plan must satisfy. In this case, we use the synthetic globally operator $G\phi$, which describes that the state formula $\phi$ should always hold. For more details on PCTL formulas see, e.g., [5, 12]. In this manner, the safety constraint to avoid location ⑤ is described by the formula $G(\neg ⑤)$.

*Bounding cost functions:* In this case, we restrict policies based on their final expected cumulative reward, for example, constraining the robot to arrive at its final location within a certain time limit. Only policies that satisfy this bound constraint will be considered, while the ones that violate it will be discarded. Formally, we can use the reward operator $\mathcal{R}^{\rho}_{\sim r}[C]$ to specify these constraints. Bounding cost functions can be used to express bounding measures constraints. For example, if we want to bound the travel time of the robot to be less than 5 time units, then we introduce the constraint $\mathcal{R}^{c_t}_{\leq 5}[C]$, where $c_t$ is the travel time cost function.

*4.3.3 Combining preferences and constraints:* So far, we have described how to define preferences and constraints using PCTL formulas. However, these formulas were defined separately and hence can not be directly combined. For example, we cannot combine a PCTL formula with a probability operator and one with a reward operator. For this reason, we combine all the constraints using multi-objective definition [13]. In our robot planning example, we define the following multi-objective goal: $multi(\mathcal{R}^{c}_{\min}[C], \mathcal{P}_{\geq 1}[F⑥ \wedge G(\neg ⑤)], \mathcal{R}^{c_t}_{\leq 5}[C])$. The first objective is to minimize the multi-objective cost function $c$. Next, we want to reach state ⑥ while avoiding ⑤ with probability 1. Finally, the last objective states that the expected travel time should be less than or equal to 5. Generally, the multi-objective goal is defined as: $multi(\mathcal{R}^{c}_{\min}[C], \mathcal{P}_{\geq 1}[F\phi_{goal} \wedge \phi_{safe}], \text{bounding}_1, \ldots, \text{bounding}_n)$, where $\phi_{goal}$ defines the reachability goal, $\phi_{safe}$ defines the safety constraint, and each $\text{bounding}_i$ defines a bounding constraint. Note that the multi-objective goal has only one optimization task, i.e., $\mathcal{R}^{c}_{\min}[C]$.

## 4.4 Policy Synthesizer

The Policy Synthesizer executes the MDP based on the PCTL property defined in the previous step. It outputs a mission policy that can be translated into a sequence of actions to be executed by a self-adaptive system.

The output from the Policy Synthesizer is also used to calculate the policy's utility. In Section 2, we defined the utility function of a policy using the cost functions to describe how "bad" the cost of $p$ is in comparison to the maximum cost for any possible policy. In this paper, we define it as $u_i(p) = 1 - c_i(p)/\max_{o \in \Pi}\{c_i(o)\}$, although for other contexts different cost functions might be more appropriate. To avoid policies with infinite cost, we assume that the graph representation of the map does not have any loops. We compute the maximal cost for each cost function using the PCTL formula $\mathrm{R}^{c_i}_{\max}[F\phi]$, where $\phi$ defines the goal states. The overall utility of a policy is then calculated using utility function weights $w_i$ as: $u(p) = \sum_{i \in QA} w_i u_i(p)$. Lastly, given a policy $p$, we can also calculate the cost $c_i(p)$ for each cost function using the MDP model.

## 4.5 Reasoning Engine

Our approach uses a rule-based reasoning engine to analyze data in the knowledge base and create a utility function and set of constraints. The advantage of using a reasoning engine is that rules can be adjusted to the specific contexts of a system. Being based on rules, our approach can provide certain guarantees of how utility functions and constraints will be adjusted in a given situation.

We use Drools [33] as a reasoning engine. It is triggered by the Quality Attribute Adapter and operates on the knowledge base. Rules can be specified to automatically adjust utility functions (e.g., to increase the weight of privacy when a stakeholder enters the map). Drools rules can also be specified to deal with constraints. Conflicts between pairs of constraints might occur, implying that not both can be fulfilled at the same time. For conflicts between *hard* constraints, it is not possible to automatically compute a resolution, but stakeholder input needs to be collected to decide which constraints should hold.

**Listing 1: Example rule to handle conflicting constraints**

```
rule resolveConflict
    salience 5
    when
     $cons: Constraint($myQA: getQA(),
         isLowerBound(),  $myValue: getValue())
     $otherCons: Constraint(
         getQA()==$myQA, isUpperBound(),
         getValue() < $myValue,
         $otherValue : getValue())
    then
     Constraint $new = new Constraint($myQA,
         ($myValue + $otherValue) / 2)
     $new.setEqual()
     insert($new)
     addConflictTrace($new,$cons)
     addConflictTrace($new,$otherCons)
end
```

Listing 1 shows an example Drools rule to deal with conflicts between *soft* constraints. The conflict occurs if a soft constraint `$cons` restricts the lower bound of a quality attribute measure to a value higher than the upper-bound value of another soft constraint `$otherCons`. In such a case, the rule creates a new constraint that requires the measure to be equal to the mean value of the two conflicting constraints. A trace link is created to the two previous constraints and the system can resume planning without any constraint conflicts.

In practice, there are multiple ways of dealing with conflicting constraints, and choosing the mean might only be a valid default strategy for soft constraints. Other rules (e.g., to keep constraints based on their priorities or the authority levels of stakeholders) can be added to arrive at different conflict resolution behavior, depending on the current context and stakeholder needs. Fallback strategies can be specified to deal with unknown contexts.

## 5 PRELIMINARY EVALUATION

In order to assess the applicability of our approach and to evaluate the potential benefit of adapting constraints and utility functions, we conducted a preliminary evaluation using our previously mentioned example scenario. We focus on the adaptation of constraints and utility functions using the Quality Attribute Adapter and the utilities of the generated policies.

### 5.1 Evaluation Setup

We use the example described in Section 2 of a warehouse in which a robot performs tasks that include traveling from the start location, visiting a location (e.g., to collect items), and returning to its initial location. We use the map shown in Figure 1. Although our framework supports MDP models in general, for the sake of simplicity of our explanations, we present an example with a deterministic MDP (without probabilities).

For the experiments, we calculated the (1) obtained utility values, (2) costs, and (3) the generated policies with and without adaptation considering different utility function weights and constraints. As quality attributes we selected *safety*, which describes the number of expected collisions, the *completion time of the plan*, and *privacy* (i.e., non-intrusiveness of humans' personal spaces), which is measured

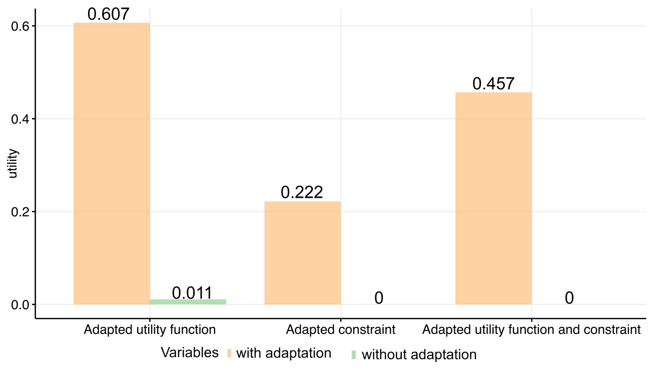

**Figure 3: Initial results with/without QA adaptation**

by the number of traversed private or semi-private locations. While design-time calculations can be used to verify the experimental results, the actual quality attribute adaptation in our approach is performed at run time.

For policy synthesis, we used the off-the-shelf probabilistic model checker PRISM[1] [25] to generate plans for a robot to travel from ① to ⑥ using the following utility function weights and constraints:

(1) Equal weights for all three quality attributes (0.333) and no constraints (for the planner without adaptation);
(2) Adjusted weights to support the delivery of a valuable/dangerous item: 0.9 for $w_{safe}$, 0.1 for $w_{time}$, 0 for $w_{priv}$ (for the planner with an adapted utility function)
(3) A human enters the map and makes location ⑤ unavailable to travel through (for the planner with an adapted constraint)

### 5.2 Evaluation Results

This section presents the results of our preliminary evaluation.

Figure 3 shows the obtained utility for the example scenario. The results stem from the cost and utility calculations in Table 1. It is

[1]http://www.prismmodelchecker.org

**Table 1: Cost and utilities of different plans depending on utility functions and constraints**

| $w_{safe}$ | $w_{time}$ | $w_{priv}$ | con-straint | Plan (optimal) | $c_{safe}$ | $c_{time}$ | $c_{priv}$ | $u_{safe}$ | $u_{time}$ | $u_{priv}$ | $u$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.333 | 0.333 | 0.333 | no | 2, 3 | 1 | 3.414 | 3 | 0.667 | 0.065 | 0 | 0.243 |
| | | | | 4, 3 | 2 | 3.65 | 1 | 0.333 | 0 | 0.667 | 0.333 |
| | | | | 4, 5 (✓) | 3 | 3.236 | 0 | 0 | 0.113 | 1 | 0.371 |
| 0.333 | 0.333 | 0.333 | yes | 2, 3 | 1 | 3.414 | 3 | 0.5 | 0.065 | 0 | 0.188 |
| | | | | 4, 3 (✓) | 2 | 3.65 | 1 | 0 | 0 | 0.667 | 0.222 |
| | | | | 4, 5 | violates constraint (forbid location 5) | | | | | | 0 |
| 0.9 | 0.1 | 0 | no | 2, 3 (✓) | 1 | 3.414 | 3 | 0.667 | 0.065 | 0 | 0.607 |
| | | | | 4, 3 | 2 | 3.65 | 1 | 0.333 | 0 | 0.667 | 0.3 |
| | | | | 4, 5 | 3 | 3.236 | 0 | 0 | 0.113 | 1 | 0.011 |
| 0.9 | 0.1 | 0 | yes | 2, 3 (✓) | 1 | 3.414 | 3 | 0.5 | 0.065 | 0 | 0.457 |
| | | | | 4, 3 | 2 | 3.65 | 1 | 0 | 0 | 0.667 | 0 |
| | | | | 4, 5 | violates constraint (forbid location 5) | | | | | | 0 |

shown how the utility changes depending on whether the utility function, constraints, or both are adapted, or whether no adaptation is used. In the table, the utility function weights $w_i$ are shown, along with whether or not the example constraint (of forbidding ⑤) is considered. For each plan, it is indicated whether it is optimal and the cost and utilities in different quality attribute dimensions are shown ($c_i$ for costs, $u_i$ for utilities). In all columns, $safe$ stands for safety, $time$ for the travel time, and $priv$ for privacy. Finally, the total utility $u$ is shown in the rightmost column, which is the weighted sum of the utilities of different quality attributes.

For our evaluation example, we assume all initial utility function weights to be 0.333 in the no-adaptation system (leading to a fixed policy via ④ and ⑤, which is used as the plan without adaptation). In the scenario of the task change that requires safety to be prioritized higher and the weights to be adjusted to (0.9, 0.1, 0), the policy via ② and ③ is selected, leading to a utility of 0.607. On the other hand, the no-adaptation plan has a utility of 0.011 (as it would still select the policy via ④ and ⑤). When considering the example constraint (i.e., making location ⑤ unavailable), the no-adaptation system would be unable to plan for it and arrive at a utility of 0. Note that because the plan via ④ and ⑤ is not usable anymore, the utilities of the quality attributes need to be recalculated, given that the maximum cost might have changed. For instance, while the maximum cost of collision was 3 for the no-constraint plans, it is 2 now that the plan via ④ and ⑤ is disregarded, which impacts the obtained utilities for the two other plans.

In the case of constraint adaptation, it is possible for our adaptive framework to plan a policy by avoiding location ⑤ and achieve a utility of 0.222 by choosing the path via ④ and ③ (assuming that all utility function weights are 0.333). In case both the utility function and constraint are adapted, the utility of the adapted system is 0.457 (in comparison to 0 for the no-adaptation system).

## 6 DISCUSSION AND RESEARCH OUTLOOK

Results from our preliminary evaluation have shown that adapting quality attributes can provide plans with higher utilities. Depending on the concrete system and mission in a specific context, the utility-focused evaluation can result in more or less substantial findings. In our case, the difference was highest (i.e., 0.596) for the scenario in which only the utility function needed to be adapted.

To further evaluate our approach, we plan to run experiments with real-world systems and assess the required effort when adapting quality attributes. Our approach involves the re-construction of the model and the synthesis of a new policy at run time, which can induce a high overhead. To address this issue, hybrid planning [31] or plan reuse [23] can be leveraged to reduce the cost of replanning, react as quickly as needed, and potentially reuse pre-computed plans. In certain situations, for example, if a location is only unavailable during a limited time interval, it might not be beneficial to replan the policy and adapt quality attributes. The goal should be to have a sufficiently stable policy in most contexts and only replan when needed.

Moreover, our approach does not need to be based on policy synthesis using MDPs but could also employ other automated planning techniques. We envision our approach to be applicable to diverse self-adaptive systems with MAPE-K components. As an integrated part of a system, our approach can inform the design of these components, e.g., to define what monitoring data should be collected to trigger quality-related changes, or how and when re-planning should occur.

With our initial prototype and application example, we have demonstrated that our approach can be leveraged to increase the utility obtained by selected plans, considering changed utility functions and constraints. Based on these initial results, we are planning to extend our work and specifically target four main areas:

**Diverse types of constraints:** Up until now our approach supports defining constraints that limit the measures of a quality attribute or restrict the proximity or speed of self-adaptive systems. Additional types of constraints to support are, for instance, temporal logic constraints (so that time-bound constraints can be expressed). Conflicts between different types of constraints might need to be resolved with different strategies. For hard constraints, it is not feasible to resolve conflicts automatically, and stakeholder input needs to be collected.

**Different types of preference representations:** While we focus on weighted sum utility functions in this paper, the approach can be extended to support other kinds of utility functions (e.g., weighted products [42]) by adjusting the Model Creator and Policy Synthesizer. However, in practice, it can be a non-trivial task to define utility functions. Certain systems and contexts might require different preference representations. For instance, it can be desirable to compute "knee points", which are well-balanced trade-offs between several objectives [6, 16]. Future work can incorporate this notion, so that knee-point solutions are selected by default and alternative solutions can be chosen when priorities of quality attributes change.

**Support for stakeholder input and explainability:** In our envisioned approach, stakeholders should be able to indicate preferences and constraints at run time. Understanding the adaptation behavior of the system is key to make appropriate decisions. Future work will examine what the Quality Attribute Configurator should look like, how adaptation actions can be explained, and how decision support can be provided to assist stakeholders when giving input, resolving conflicts, and making decisions. Mechanisms can be added to elicit preferences and constraints for previously unconsidered quality attributes. To deal with scalability issues, we aim to limit the required stakeholder input by initially eliciting default preferences and constraints [46] and adjusting them only in specific situations. Our approach is semi-automatic and involves stakeholders only when needed (e.g., when dealing with hard or safety-critical conflicting constraints).

**Evaluation of the approach:** To evaluate the proposed approach, we intend to perform a human subjects study. The focus will lie on the ease of use and the understandability of our approach. The study can be performed as a think-aloud study (to elicit participants' mental models while working with the tool/approach). Additionally, controlled experiments can be conducted to understand whether the provided explanations can help humans make decisions more confidently and deliberately in comparison to users that select utility functions without any guidance.

## 7 RELATED WORK

Our approach addresses the need to manage requirements for self-adaptive systems, deal with uncertainty, and involve users in interactive decision-making at run time [4]. Relevant related work is concerned with the run-time adaptation of utility functions, constraints, and goal models.

*Adjusting utility functions at run time:* While utility functions have been widely used for self-adaptive systems [8, 9, 11, 15, 17, 40], only in recent years, the need has been raised to re-adjust utility functions at run time [22, 27] based on changing user preferences [27]. One approach that focuses on this issue switches between "variants" of utility functions depending on the system's context [21]. Another approach uses fuzzy logic to adapt utility functions based on predefined adaptation rules [3]. Instead of requiring stakeholders to describe rules for all possible contexts at design time, our approach supports user input at run time.

The topic of utility function adaptation is similar to recent work on adjusting priorities at run time [36], which uses ML techniques to adjust priorities of quality attributes (which are similar to our utility function weights) to ensure that QoS constraints are met.

*Adjusting constraints at run time:* Several tools have been proposed to detect and resolve conflicts between requirements, e.g., the Oz System that can automatically detect conflicts and find compromise solutions [34]. For self-adaptive systems, Song et al.'s approach elicits end user preferences and constraints, which are used to find a solution for a constraint satisfaction problem that minimizes the number of violated goals [39].

The language RELAX allows stakeholders to specify requirements for self-adaptive systems under uncertainty and supports several operators to indicate how a requirement can be relaxed at run time [45]. The issue of conflicting requirements is mentioned, along with the potential use of temporal constraint analysis to identify inconsistent pairs of RELAX constraints [38].

In the context of smart cities, a decision support system has been developed that uses Integer Linear Programming to resolve conflicts between constraints [28]. Our work is similar in the sense that it collects input from stakeholders at run time and supports the semi-automatic resolution of constraints.

*Goal-oriented self-adaptation:* Several approaches have applied goal modeling in the context of self-adaptive systems [2, 14, 30, 35]. For example, FLAGS [2] uses KAOS and LTL to support the run-time adaptation of goals. ActivFORMS [18] supports goal model adaptation at run time and uses the Uppaal model checker for checking TCTL expressions. While ActivFORMS allows to change and verify goals at run time, our work is more strongly focused on quality attributes and provides mechanisms for conflict resolution. Another approach combines KAOS and RELAX to identify and mitigate uncertainty factors in requirements for self-adaptive systems [7]. In the domain of CPS architecture adaptation, an approach [14] for architectural self-adaptation has been developed based on goal models and predictive monitoring to deal with operational uncertainty. While these approaches tackle important aspects of self-adaptive systems, they do not focus on the adaptation of quality attributes based on run-time user input and conflict resolution.

The issue of conflicts in goal-oriented requirements engineering has also been studied, along with solution strategies (e.g., goal weakening or resolution heuristics) [43]. We plan to build upon that work to inform the development of conflict resolution mechanisms.

## 8 CONCLUSION

In this paper, we have presented an initial approach for the dynamic adaptation of quality attributes for automated planning. Our approach supports the semi-automatic adjustment of utility functions and constraints, including support to process input from multiple stakeholders and resolve conflicts between soft constraints. Our evaluation indicated that the approach can lead to higher utility values in comparison to approaches that do not support the adaptation of utility functions and constraints. We presented a research outlook that includes directions for future work, such as supporting different types of constraints and preference representations, creating comprehensible interfaces, and evaluating the approach. As part of our ongoing work, we are implementing a system to support the described approach, designing an empirical study for evaluation, and developing a comprehensive user interface to elicit stakeholder input and support explainability.

## ACKNOWLEDGMENTS

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, and the Linz Institute of Technology (LIT-2019-7-INC-316).

## REFERENCES

[1] Jonathan Aldrich, David Garlan, et al. 2019. Model-based adaptation for robotics software. *IEEE Software* 36, 2 (2019), 83–90.

[2] Luciano Baresi, Liliana Pasquale, and Paola Spoletini. 2010. Fuzzy goals for requirements-driven adaptation. In *Proceedings of the 18th International Requirements Engineering Conference*. IEEE, 125–134.

[3] Mounir Beggas, Lionel Médini, Frederique Laforest, and Mohamed Tayeb Laskri. 2013. *Fuzzy Logic Based Utility Function for Context-Aware Adaptation Planning*. Springer International Publishing, Cham, 227–236.

[4] Nelly Bencomo, Jon Whittle, Pete Sawyer, Anthony Finkelstein, and Emmanuel Letier. 2010. Requirements reflection: requirements as runtime entities. In *Proceedings of the ACM/IEEE 32nd International Conference on Software Engineering*, Vol. 2. ACM, New York, 199–202. https://doi.org/10.1145/1810295.1810329

[5] Andrea Bianco and Luca de Alfaro. 1995. Model checking of probabilistic and nondeterministic systems. In *Foundations of Software Technology and Theoretical Computer Science*, P. S. Thiagarajan (Ed.). Springer Berlin Heidelberg, 499–513.

[6] Tao Chen, Ke Li, Rami Bahsoon, and Xin Yao. 2018. FEMOSAA: Feature-Guided and Knee-Driven Multi-Objective Optimization for Self-Adaptive Software. *ACM Transactions of Software Engineering and Methodology* 27, 2, Article 5 (jun 2018). https://doi.org/10.1145/3204459

[7] Betty H. C. Cheng, Pete Sawyer, Nelly Bencomo, and Jon Whittle. 2009. A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems*, Andy Schürr and Bran Selic (Eds.). Springer Berlin Heidelberg, 468–483.

[8] Shang-Wen Cheng, David Garlan, and Bradley Schmerl. 2006. Architecture-based self-adaptation in the presence of multiple objectives. In *Proceedings of the International Workshop on Self-Adaptation and Self-Managing Systems*. ACM, New York, 2–8. https://doi.org/10.1145/1137677.1137679

[9] Javier Cámara, Antónia Lopes, David Garlan, and Bradley Schmerl. 2016. Adaptation impact and environment models for architecture-based self-adaptive systems. *Science of Computer Programming* 127 (2016), 50–75.

[10] Naeem Esfahani, Ahmed Elkhodary, and Sam Malek. 2013. A learning-based framework for engineering feature-oriented self-adaptive software systems. *IEEE Transactions on Software Engineering* 39, 11 (2013), 1467–1493.

[11] Funmilade Faniyi, Peter R. Lewis, Rami Bahsoon, and Xin Yao. 2014. Architecting Self-Aware Software Systems. In *Proceedings of the 2014 IEEE/IFIP Conference on Software Architecture*. IEEE, 91–94. https://doi.org/10.1109/WICSA.2014.18

[12] Vojtěch Forejt, Marta Kwiatkowska, Gethin Norman, and David Parker. 2011. *Automated Verification Techniques for Probabilistic Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 53–113.

[13] V. Forejt, M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. 2011. Quantitative Multi-Objective Verification for Probabilistic Systems. In *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'11)*. Springer, 112–127.

[14] Ilias Gerostathopoulos, Tomas Bures, Petr Hnetynka, Jaroslav Keznikl, Michal Kit, Frantisek Plasil, and Noël Plouzeau. 2016. Self-adaptation in software-intensive cyber-physical systems: From system goals to architecture configurations. *Journal of Systems and Software* 122 (2016), 378–397.

[15] Carlo Ghezzi and Amir Molzam Sharifloo. 2013. Dealing with Non-Functional Requirements for Adaptive Systems via Dynamic Software Product-Lines. In *Software Engineering for Self-Adaptive Systems II*, Rogério de Lemos, Holger Giese, Hausi A. Müller, and Mary Shaw (Eds.). Springer Berlin Heidelberg, 191–213.

[16] Sara Hassan, Nelly Bencomo, and Rami Bahsoon. 2015. Minimizing Nasty Surprises with Better Informed Decision-Making in Self-Adaptive Systems. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2015)*. IEEE, 134–145. https://doi.org/10.1109/SEAMS.2015.13

[17] William Heaven, Daniel Sykes, Jeff Magee, and Jeff Kramer. 2009. A Case Study in Goal-Driven Architectural Adaptation. In *Software Engineering for Self-Adaptive Systems*. Springer Berlin Heidelberg, 109–127.

[18] M Usman Iftikhar and Danny Weyns. 2014. ActivFORMS: Active formal models for self-adaptation. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2014)*. ACM, New York, 125–134.

[19] Paola Inverardi and Marco Mori. 2013. A software lifecycle process to support consistent evolutions. In *Self-Adaptive Systems*, R. de Lemos (Ed.). Vol. 7475 LNCS. Springer Berlin Heidelberg, 239–264.

[20] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. 1998. Planning and acting in partially observable stochastic domains. *Artificial intelligence* 101, 1-2 (1998), 99–134.

[21] Konstantinos Kakousis, Nearchos Paspallis, and George Papadopoulos. 2008. Optimizing the Utility Function-Based Self-adaptive Behavior of Context-Aware Systems Using User Feedback. In *Proceedings of the OTM Confederated International Conferences*. 657–674.

[22] Jeffrey Kephart. 2021. Viewing Autonomic Computing through the Lens of Embodied Artificial Intelligence: A Self-Debate. Keynote at the 16th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2021).

[23] Cody Kinneer, Zack Coker, Jiacheng Wang, David Garlan, and Claire Le Goues. 2018. Managing uncertainty in self-adaptive systems with plan reuse and stochastic search. In *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 40–50.

[24] H.S. Kushner. 1971. *Introduction to Stochastic Control*. Holt, Rinehart and Winston.

[25] M. Kwiatkowska, G. Norman, and D. Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11) (LNCS, Vol. 6806)*. Springer, 585–591.

[26] L. Lamport. 1977. Proving the Correctness of Multiprocess Programs. *IEEE Transactions on Software Engineering* SE-3, 2 (1977), 125–143. https://doi.org/10.1109/TSE.1977.229904

[27] Veronika Lesch, Marius Hadry, Samuel Kounev, and Christian Krupitzer. 2021. Utility-based Vehicle Routing Integrating User Preferences. In *Proceedings of the 2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events*. IEEE, 263–268.

[28] Meiyi Ma, John A. Stankovic, and Lu Feng. 2018. Cityresolver: A Decision Support System for Conflict Resolution in Smart Cities. In *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems* (Porto, Portugal) *(ICCPS '18)*. IEEE Press, 55–64. https://doi.org/10.1109/ICCPS.2018.00014

[29] Thomas L. McCluskey, Mauro Vallati, and Santiago Franco. 2017. Automated Planning for Urban Traffic Management. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*. AAAI Press, 5238–5240.

[30] Mirko Morandini, Loris Penserini, Anna Perini, and Alessandro Marchetto. 2017. Engineering requirements for adaptive systems. *Requirements Engineering* 22, 1 (2017), 77–103.

[31] Ashutosh Pandey, Gabriel A Moreno, Javier Cámara, and David Garlan. 2016. Hybrid planning for decision making in self-adaptive systems. In *Proceedings of the 10th International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE, 130–139.

[32] Simon Parkinson, Andrew Longstaff, Andrew Crampton, and Peter Gregory. 2012. The application of automated planning to machine tool calibration. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling*.

[33] Mark Proctor. 2011. Drools: a rule engine for complex event processing. In *Proceedings of the International Symposium on Applications of Graph Transformations with Industrial Relevance*. Springer, 2–2.

[34] William N Robinson and Stephen Fickas. 1994. Supporting multi-perspective requirements engineering. In *Proceedings of the IEEE International Conference on Requirements Engineering*. IEEE, 206–215.

[35] Davide Rossi, Francesco Poggi, and Paolo Ciancarini. 2018. Dynamic high-level requirements in self-adaptive systems. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. ACM, New York, 128–137.

[36] Huma Samin, Nelly Bencomo, and Peter Sawyer. 2021. Pri-AwaRE: Tool Support for priority-aware decision-making under uncertainty. In *Proceedings of the 29th International Requirements Engineering Conference (RE'21)*. IEEE, 450–451.

[37] Pete Sawyer, Nelly Bencomo, Jon Whittle, Emmanuel Letier, and Anthony Finkelstein. 2010. Requirements-Aware Systems: A Research Agenda for RE for Self-adaptive Systems. In *Proceedings of the 18th International Requirements Engineering Conference (RE'10)*. IEEE, 95–103. https://doi.org/10.1109/RE.2010.21

[38] Eddie Schwalb and Lluís Vila. 1998. Temporal constraints: A survey. *Constraints* 3, 2 (1998), 129–149.

[39] Hui Song, Stephen Barrett, Aidan Clarke, and Siobhán Clarke. 2013. Self-adaptation with End-User Preferences: Using Run-Time Models and Constraint Solving. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MODELS'13)*. Springer Berlin Heidelberg, 555–571.

[40] João Pedro Sousa, Rajesh Krishna Balan, Vahe Poladian, David Garlan, and Mahadev Satyanarayanan. 2008. User guidance of resource-adaptive systems. In *Proceedings of the 3rd International Conference on Software and Data Technologies*. 36–44.

[41] Roykrong Sukkerd, Reid Simmons, and David Garlan. 2018. Towards explainable multi-objective probabilistic planning. In *Proceedings of the 4th International Workshop on Software Engineering for Smart Cyber-Physical Systems*. ACM, 19–25. https://doi.org/10.1145/3196478.3196488

[42] Evangelos Triantaphyllou. 2000. *Multi-Criteria Decision Making Methods*. Springer US, Boston, MA, 5–21.

[43] Axel Van Lamsweerde, Robert Darimont, and Emmanuel Letier. 1998. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering* 24, 11 (1998), 908–926.

[44] Danny Weyns and Radu Calinescu. 2015. Tele Assistance: A Self-Adaptive Service-Based System Exemplar. In *Proceedings of the IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 88–92. https://doi.org/10.1109/SEAMS.2015.27

[45] Jon Whittle, Pete Sawyer, Nelly Bencomo, Betty HC Cheng, and Jean-Michel Bruel. 2010. RELAX: a language to address uncertainty in self-adaptive systems requirement. *Requirements Engineering* 15, 2 (2010), 177–196.

[46] Rebekka Wohlrab and David Garlan. 2021. A Negotiation Support System for Defining Utility Functions for Multi-Stakeholder Self-Adaptive Systems. *Requirements Engineering* (2021).