The Journal of Systems & Software 198 (2023) 111538

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Explaining quality attribute tradeoffs in automated planning for self-adaptive systems☆

Rebekka Wohlrab [a,b,*], Javier Cámara [c], David Garlan [b], Bradley Schmerl [b]

[a] Department of Computer Science and Engineering, Chalmers | University of Gothenburg, Gothenburg, Sweden
[b] Software and Societal Systems Department, Carnegie Mellon University, Pittsburgh, USA
[c] ITIS Software, University of Málaga, Málaga, Spain

ARTICLE INFO

Article history:
Received 31 May 2022
Received in revised form 2 September 2022
Accepted 6 October 2022
Available online 13 October 2022

Keywords:
Explainable software
Automated planning
Self-adaptation
Quality attributes
Non-functional requirements
Principal component analysis
Clustering
Decision tree learning

ABSTRACT

Self-adaptive systems commonly operate in heterogeneous contexts and need to consider multiple quality attributes. Human stakeholders often express their quality preferences by defining utility functions, which are used by self-adaptive systems to automatically generate adaptation plans. However, the adaptation space of realistic systems is large and it is obscure how utility functions impact the generated adaptation behavior, as well as structural, behavioral, and quality constraints. Moreover, human stakeholders are often not aware of the underlying tradeoffs between quality attributes. To address this issue, we present an approach that uses machine learning techniques (dimensionality reduction, clustering, and decision tree learning) to explain the reasoning behind automated planning. Our approach focuses on the tradeoffs between quality attributes and how the choice of weights in utility functions results in different plans being generated. We help humans understand quality attribute tradeoffs, identify key decisions in adaptation behavior, and explore how differences in utility functions result in different adaptation alternatives. We present two systems to demonstrate the approach's applicability and consider its potential application to 24 exemplar self-adaptive systems. Moreover, we describe our assessment of the tradeoff between the information reduction and the amount of explained variance retained by the results obtained with our approach.

© 2022 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Self-adaptive systems are becoming increasingly used in a variety of contexts and need to consider multiple quality attributes (e.g., security, safety, energy consumption, cost, and performance). An important step for these systems is to *plan* adaptations to changes in their environment at run time.

Consider, for instance, a robotic system that performs a variety of tasks (e.g., delivering items, visiting a number of locations, or assisting humans in their work). In certain situations, speed might be extremely relevant, whereas other contexts might require the system to be energy-efficient and safe. When planning adaptation behavior, quality-related preferences and constraints need to be taken into account, which often requires making tradeoffs among quality attributes. For instance, a robotics system cannot travel at a high speed and be extremely energy-efficient at the same time, which is why stakeholders need to prioritize these quality attributes and possibly specify constraints (e.g., to prohibit the robot from going below a certain battery level).

Quality-related preferences and constraints are often encoded in a *utility function* whose value should be maximized by the system. Multiple approaches to engineering self-adaptation employ utility functions to capture user preferences and constraints, which are often defined as weighted sums of objectives (Garlan et al., 2004; Kounev et al., 2014). These utility functions are used in the planning process, which is commonly facilitated by automated planning techniques, e.g., using probabilistic planning (Lacerda et al., 2014; García et al., 2019; Cámara et al., 2020; Jamshidi et al., 2019). Given an initial state, a set of possible actions, intermediate states, and a utility function, these techniques compute *policies* which indicate the sequences of actions a system should perform to arrive at a goal state. When performing multi-objective policy synthesis, e.g., using probabilistic model checking, it is often the case that a number of Pareto-optimal solutions exist, in which all policies are known to meet the imposed quantitative and behavioral constraints. Among those Pareto-optimal solutions, one solution has to be selected that optimizes the objectives in the most appropriate manner for a given situation/context. To define a utility function that can be used to select a solution, human preferences are needed (Wohlrab

☆ Editor: Juan Carlos Duan.

* Corresponding author at: Department of Computer Science and Engineering, Chalmers | University of Gothenburg, Gothenburg, Sweden.
E-mail address: wohlrab@chalmers.se (R. Wohlrab).

https://doi.org/10.1016/j.jss.2022.111538
0164-1212/© 2022 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

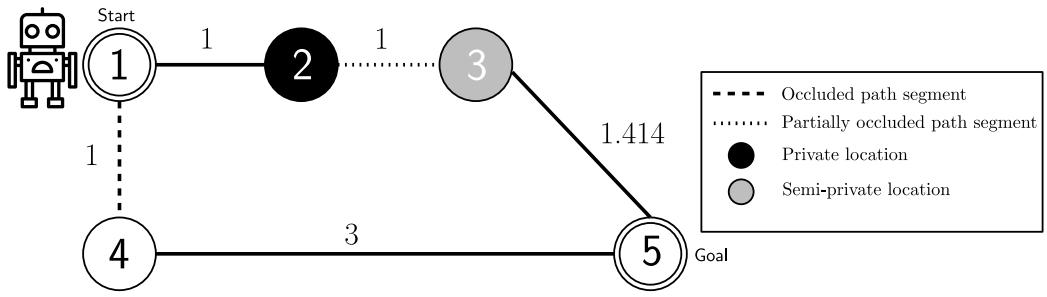


Fig. 1. Example of a robot mission planning scenario.

and Garlan, 2021b). For realistic systems, however, the state and action spaces are typically large (and may contain hundreds of thousands of states Chen et al., 2020), which makes it difficult for human users to verify that a utility function correctly captures their needs and leads to desirable policies being generated.

Human decision makers need tools and techniques to help them understand the tradeoffs of complex decision spaces and guide them to good utility functions and preference specifications, enabling them to answer questions such as: Why are these policies being generated, and not others? What are the underlying tradeoffs among quality attributes? How sensitive to changes in prioritization is the satisfaction of constraints or the achievement of optimality? Which are the key choices that drive the most important changes in adaptation behavior? What categories of adaptation behavior exist and what are their characteristics? What changes in the utility function would lead to different kinds of adaptation behavior being generated?

When aiming to address these questions, identifying the important characteristics of planning is similar to the common machine learning problem of ascertaining the distinguishing features of high-dimensional spaces. Therefore, we leverage machine learning techniques in our approach to explain the underlying reasoning behind automated planning with a focus on quality-related concerns. Our approach is based on a combination of machine learning techniques: principal component analysis (PCA) (Jolliffe, 1986), multiple correspondence analysis (MCA) (Le Roux and Rouanet, 2009), k-means clustering (van de Velden et al., 2019; Markos et al., 2019), and decision tree learning (DTL) (Breiman et al., 2017). These methods are complementary: PCA and MCA are dimensionality reduction techniques that can be useful in explaining variances in large datasets; k-means clustering is useful to investigate patterns/clusters in the generated policies; and DTL is a method (broadly employed in data mining) that helps to predict the value of a target variable based on one or more input variables. As we argue in this paper, the combination of these techniques can help human stakeholders to understand how differences in utility functions might result in differences in the adaptation behavior being generated.

We present two example systems and scenarios to demonstrate the applicability of our approach, as well as provide guidelines for human decision makers aiming to leverage the approach. To indicate the required effort of applying our approach to other systems, we discuss the necessary data collection and preparation steps for 24 exemplars for self-adaptive systems.

The remainder of this paper is structured as follows: Section 2 presents an example of automated planning to motivate the importance of explaining quality attribute tradeoffs. Section 4 introduces our approach, Section 5 describes how we leverage the selected machine learning techniques, and 6.2 describes our evaluation. We present related work in Section 7 and a discussion of our findings in Section 8. We conclude this paper in Section 9.

2. Example: Robot mission planning

Although our technique applies to a large range of self-adaptive systems, the example we consider in this paper is a robot mission planning scenario, in which a robot needs to find an optimal navigation path from a start location to a goal location. We selected this example since automated planning is frequently used in the robotics domain and it is crucial that users understand robot mission plans to facilitate human–robot interaction (Chen et al., 2020; Sukkerd et al., 2020). Fig. 1 shows an illustration of this scenario. The robot is currently at its start location ① and should move to the goal location ⑤. It can choose several paths and drive through multiple locations, some of which are private (②) or semi-private (③). Moreover, some paths between locations are occluded or partially occluded by obstacles (e.g., the path between ② and ③ is partially occluded). The robot should arrive as quickly as possible, avoid collisions, and be as unintrusive as possible (i.e., due to privacy concerns avoid areas that are occupied by humans). The robot is aware of its speed and location on the map. It has sensors to detect collisions with obstacles, and can adjust its speed (which can also reduce the probability of collisions) and select the path it should follow. In this example, a policy consists of state–action pairs that indicate how a robot would move from the initial state to the goal state. Each state consists of the robot's location and speed. Actions can either result in a changed speed (`setSpeed(...)`) or in a changed target location (`moveTo(...)`). For the sake of simplicity, we consider a fixed speed of 1 in this example. We define the utility function as a weighted sum of relevant objectives whose costs should be reduced (i.e., travel time, collisions, and intrusiveness) (Ghezzi and Molzam Sharifloo, 2013; Cheng et al., 2006; Esfahani et al., 2013; Sousa et al., 2008). In this case, the utility function for a policy σ would be:

$$u(\sigma) = w_t \cdot u_t(\phi_t(\sigma)) + w_c \cdot u_c(\phi_c(\sigma)) + w_i \cdot u_i(\phi_i(\sigma)) \quad (1)$$

The utility function of σ is the weighted sum (Triantaphyllou, 2000) of the utilities for travel time (u_t), collisions (u_c), and intrusiveness (u_i). Each of the individual utility functions u_* map the domain of the corresponding variables to the range [0, 1]. In the expression, ϕ_* denotes functions that return the quantified travel time, collisions, and intrusiveness of σ, respectively. Weights w_* indicate the importance of the quality attributes and sum to 1.

To illustrate the difficulty of performing quality attribute tradeoffs and understanding utility functions, we can calculate the costs (of travel time, collisions, and intrusiveness) and their respective utilities in our example planning problem. Table 1 shows an example of the alternative paths (either via ② and ③ or via ④), their costs, and utilities depending on the utility function weights. For this example, we assume that the cost of travel time $\phi_t(\sigma)$ is equivalent to the distance between locations (the distance of the horizontal and vertical path segments is assumed to be 1, the distance of the diagonal path segment between ③ and ⑤ is $\sqrt{2}$ = 1.414, and the distance between ④ and ⑤ is 3).

Table 1
Example paths and their costs depending on utility function weights.

Weights			Locations	Optimal	Costs			Utilities			
w_t	w_c	w_i			ϕ_t	ϕ_c	ϕ_i	u_t	u_c	u_i	u
1	0	0	②,③	✓	3.414	1	3	0.147	0.5	0	0.147
			④		4	2	0	0	0	1	0
0.33	0.33	0.33	②,③		3.414	1	3	0.147	0.5	0	0.215
			④	✓	4	2	0	0	0	1	0.333
0	0.5	0.5	②,③		3.414	1	3	0.147	0.5	0	0.25
			④	✓	4	2	0	0	0	1	0.5
0	1	0	②,③	✓	3.414	1	3	0.147	0.5	0	0.5
			④		4	2	0	0	0	1	0
0.5	0.5	0	②,③	✓	3.414	1	3	0.147	0.5	0	0.323
			④		4	2	0	0	0	1	0
0	0	1	②,③		3.414	1	3	0.147	0.5	0	0
			④	✓	4	2	0	0	0	1	1
0.5	0	0.5	②,③		3.414	1	3	0.147	0.5	0	0.073
			④	✓	4	2	0	0	0	1	0.5

Moreover, we assume that the expected cost $\phi_c(\sigma)$ of a collision is 1 for a partially occluded and 2 for an occluded path segment, respectively, and 0 for non-occluded path segments. The cost of intrusiveness $\phi_i(\sigma)$ is 1 for every semi-private location the robot visits and 2 for every private location that is traversed. From the start location ①, the path via ② and ③ would cost 3.414 in terms of travel time, 1 in terms of collision, and 3 in terms of intrusiveness. The path via ④ is longer than the other path (cost 4 for travel time), is occluded between ① and ④ (i.e., cost 2 in terms of collision), and is not intrusive (cost 0).

To define the local utility functions u_*, we aim to capture how far away from the 'worst' or 'most expensive' value the costs in the three dimensions are. In terms of travel time, 4 is the maximum cost when traveling from the start to the goal location. In terms of collision, the maximum cost of the possible paths is 2 (with one occluded edge). For intrusiveness, the maximum cost is 3 (given that both ② and ③ are non-public). For the local utility functions, we are interested in the ratio of our actual cost and the maximum cost in one dimension. We subtract that ratio from 1 to indicate that a lower ratio is what is desirable and leads to higher utility. We define the local utility functions in our example as follows:

$$u_t(x) = 1 - \frac{x}{4} \qquad u_c(x) = 1 - \frac{x}{2} \qquad u_i(x) = 1 - \frac{x}{3} \tag{2}$$

When using an automated planner for this example scenario, the weights of the utility function (and the prioritization of quality attributes) would determine which path is chosen. In Table 1, it can be seen that different paths would be chosen depending on the weights of travel time, collision, and intrusiveness, along with the cost. If travel time is the only important quality attribute (w_t=1), the planner would choose the path via ② and ③ (as it is shorter). In contrast, if intrusiveness is the only important quality (w_i=1), the planner would select the path via ④ as the optimal one. Clearly the preferences over quality attributes expressed in the weights have a substantial impact on the actual path that would be chosen by a planner.

In the two examples mentioned above, it is clear why the planner chose a given path as optimal in the different contexts. However, what would happen if all quality attributes are equally important ($w_* = 0.333$)? In this case, the planner would select the path via ④ in our example, but why did that happen? And does that selection really fulfill stakeholder needs?

Ideally, we would like our planner to select a non-intrusive, non-colliding path that is extremely short. However, it is unlikely that the requirements of non-intrusiveness, collision avoidance, and short travel time can all be completely fulfilled; instead, tradeoffs need to be made and it has to be decided for a specific situation what the important quality attributes are.

It can be seen that even in such a simple example, it can be difficult to reason about path planning problems and state how utility function weights should be chosen. Our approach aims to address this challenge by making the planning process and quality attribute tradeoffs more explainable to humans.

3. Automated planning with Markov decision processes

Before describing the steps of our approach, we present some concepts related to the formalism employed for automated planning in the instantiation of our approach.

The planning problem domain is described as a Markov decision process:

Definition 3.1 (*Markov Decision Process*)**.** A Markov decision process (MDP) over a set of atomic propositions AP is a tuple $\mathcal{M} = (S, s_I, A, \Delta, L, R)$, where $S \neq \emptyset$ is a finite set of states; $s_I \in S$ is the initial state; $A \neq \emptyset$ is a finite set of actions; $\Delta : S \times A \to Dist(S)$ is a partial probabilistic transition function that maps state–action pairs to discrete probability distributions over S, $L : S \to 2^{AP}$ is a labeling function that maps every state $s \in S$ to the atomic propositions from AP that hold in that state; and R is a (possibly empty) set of (reward/cost) functions $\rho : S \to \mathbb{R}_{\geq 0}$ that associate non-negative values with states of the MDP.[1]

In each state $s \in S$, the set of actions $a \in A$ for which $\Delta(s, a)$ is defined contains the actions *enabled* in state s, and is denoted by $A(s)$. The choice of which action from $A(s)$ to take in some states is assumed to be nondeterministic.

We can reason about the behavior of MDPs using *policies*. A policy resolves the nondeterministic choices included in a MDP, selecting the action to take in states where $|A(s)| > 1$. Although there are multiple classes of MDP policies, in this work, we use deterministic memoryless policies as implemented in off-the-shelf probabilistic model checkers like PRISM (Kwiatkowska et al., 2011) and Storm (Dehnert et al., 2017) (called simply 'policies' in the remainder of the paper).

Definition 3.2 (*MDP Policy*)**.** A (deterministic memoryless) policy of an MDP is a function $\sigma : S \to A$ that maps each state $s \in S$ to an action from $A(s)$.

[1] While generally, reward functions can also be associated with actions $\rho : A \to \mathbb{R}_{\geq 0}$, we consider only those that are associated with states in this paper.

To synthesize policies that satisfy constraints and optimize objective functions (defined over, e.g., quality attributes such as timeliness, safety, etc.), probabilistic model checkers can employ formulas specified in probabilistic temporal logics like PCTL (Hansson and Jonsson, 1994; Bianco and De Alfaro, 1995), which are used to quantify properties related to probabilities and rewards in system specifications modeled as MDPs.

Definition 3.3 (*PCTL Formulae*)**.** State PCTL formulae Φ and path PCTL formulae Ψ over an atomic proposition set AP are defined by the grammar:

$$\begin{aligned}\Phi &::= true \mid \alpha \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathcal{P}_{\sim p}[\Psi] \mid \mathcal{R}^{\rho}_{\sim r}[C^{\leq k}] \mid \mathcal{R}^{\rho}_{\sim r}[F\Phi] \\ \Psi &::= X\Phi \mid \Phi \ \mathrm{U} \ \Phi \mid \Phi \ \mathrm{U}^{\leq k} \ \Phi\end{aligned} \tag{3}$$

where $\alpha \in AP$ is an atomic proposition, $\sim \in \{\geq, >, <, \leq\}$ is a relational operator, $p \in [0, 1]$ is a probability bound, $r \in \mathbb{R}^{+}_{0}$ is a reward bound, $k \in \mathbb{N}_{>0}$ is a timestep bound, and $\rho \in R$ is a reward/cost function.

The semantics of the probability $\mathcal{P}$ and reward $\mathcal{R}$ operators are defined over all policies σ of M as follows: $\mathcal{P}_{\sim p}[\Psi]$ specifies that the probability that paths starting at a chosen state s satisfy a path property Ψ is $\sim p$ for all policies; $\mathcal{R}^{\rho}_{\sim r}[C^{\leq k}]$ holds if the expected cumulated reward for ρ up to time-step k is $\sim r$ for all policies; and $\mathcal{R}^{\rho}_{\sim r}[F\Phi]$ holds if the expected reward accrued for ρ before reaching a state that satisfies Φ is $\sim r$ for all policies. Replacing $\sim p$ (or $\sim r$) from (3) with $min =?$ or $max =?$ specifies that the calculation of the minimum/maximum probability (or reward) over all MDP policies is required.

In our example, we can encode, for instance, a PCTL formula that optimizes the utility associated with the timeliness of the robot's arrival at location $L5$ in the map as $\mathcal{R}^{ut}_{max=?}[FLoc = L5]$, where $ut \in R$ is a utility function for timeliness encoded as a reward/cost function, and Loc is a state variable that corresponds to the robot's location in the MDP. For a full description of the PCTL semantics, see Hansson and Jonsson (1994) and Bianco and De Alfaro (1995).

Data generation. To explore the policy space of an automated planning problem, our approach requires executing an automated planner (e.g., the probabilistic model checker PRISM) repeatedly, with different combinations of utility function weights and parameter values (i.e., using uniform distributions over the space of utility function weights).

The inputs provided to the planner are formalized under the notion of *policy exploration profile*.

Definition 3.4 (*Policy Exploration Profile*)**.** A policy exploration profile is a tuple $(\mathcal{M}, W, C_{\Phi}, U, \Pi)$, such that:

- $\mathcal{M}$ is a Markov decision process model encoding the planning domain.
- $W \subseteq \mathbb{P}([0, 1]^n)$ is a set of tuples that capture various utility weight combinations, where n is the number of utility dimensions considered. For any $w = (w_1, \ldots, w_n) \in W$, $\sum_{i=1}^{n} w_i = 1$.
- C_{Φ} is a set of PCTL formulae, each of which encodes the cost function that corresponds to a dimension of concern (e.g., timeliness, energy consumption).
- $U \subseteq \mathbb{P}(\mathbb{R}^{n}_{\geq 0} \rightarrow [0, 1]^n)$ is a set of utility functions that map costs in a given dimension of concern to the range $[0, 1]$.
- $\Pi : P \rightarrow \mathbb{P}(\mathcal{D})$ is a function that assigns sets of symbols typed by a fixed set $\mathcal{D}$ to a set of parameters P required as input to the planning problem (e.g., state variable initialization values such as the starting position of the robot).

From a policy exploration profile $\tau = (\mathcal{M}, W, C_{\Phi}, U, \Pi)$, the automated planner generates as output a set of tuples of the form $(\sigma, u_{\sigma}, u^{g}_{\sigma})$, where:

- σ is a MDP policy computed as $\sigma = f_{\sigma}(\tau.\mathcal{M}_{\pi}, \phi_c).p$, where $\pi \in \tau.\Pi$ is an assignment of parameter values for model $\mathcal{M}$, $\phi_c \in \tau.C_{\Phi}$ is a PCTL formula encoding the cost function for a given dimension of concern, and $f_{\sigma}(*).p$ designates the model checking function that synthesizes a policy from a MDP model and a PCTL property.
- $u_{\sigma} \in [0, 1]^n$ contains the utility values for the different dimensions of concern. For dimension i, its utility value is computed as $u_{\sigma i} = u_i(f_{\sigma}(\tau.\mathcal{M}_{\pi}, \phi_c).r)$, where $u_i \in U$ is the utility function for dimension i and $f_{\sigma}(*).r$ designates the model checking function that quantifies a cost function encoded as a PCTL property for a MDP model.
- u^{g}_{σ} is the global utility of policy σ, calculated as the weighted sum of utilities in u_{σ}, i.e., $u^{g}_{\sigma} = \sum_{i=1}^{n} w_i \cdot u_{\sigma i}$.

4. Approach

This section describes our approach, with a focus on how we integrate several methods to explain quality attribute tradeoffs. The approach has been designed to fulfill the requirements that we outline in Section 4.1 with the methodology and tool support described in Section 4.2. We provide a high-level description of the employed machine learning techniques in Section 4.3. Our approach relies on data filtering and extraction. We describe these steps in Section 4.4.

4.1. Requirements

This subsection describes the requirements for our approach for explaining quality tradeoffs. Challenges with automated planning stem from the large state space of the planning problem (that makes it difficult to distinguish important variables from less important ones) (Chen et al., 2020). Moreover, the large amount of possibly generated plans is hard to analyze for human stakeholders. When analyzing the actions of a plan, it is often obscure what the underlying quality tradeoffs of this plan are and what alternative actions would have been possible. It is also challenging for humans to understand how input variables (e.g., utility function weights) need to be selected to generate desirable plans. While approaches for utility function definition have been proposed in the past (Wohlrab and Garlan, 2021b), it is often unclear to human stakeholders how a defined utility function impacts the generated planning policies. Our approach for quality tradeoff explanations aims to address this issue. We developed the following list of requirements and apply a variety of machine learning techniques to fulfill them in our approach. The explainability approach should:

1. give general insights into which variables (e.g., utility function weights, costs) are important to differentiate policies and analyze their relations;
2. enable stakeholders to identify *strategies*, i.e., high-level collections of policies sharing similar characteristics;
3. help to analyze how differences in utility function weights impact the generated policies;
4. support the analysis of quantitative data (for cost and utility weight variables) and categorical data (for policy actions).

Insights into important variables and their relations (1) are needed to determine how strongly different quality attributes impact differences in policies (and if there exist any tradeoffs between

Table 2
Selection of techniques to meet different information needs.

	Category	Information need	Technique(s) in our approach
IN1	General	General understanding of variables, their contributions, and correlations	PCA, MCA
IN2	General	Exploration of categorical variables and their contributions/relationships	MCA
IN3	Key decisions	Analysis of which state–action pairs are impacted by a change to utility function weights	MCA
IN4	Strategies	Identification of clusters (collections of policies that share similar characteristics) and their key attributes	Joint k-means clustering and PCA/MCA
IN5	Utility function weights	Thresholds (e.g., in utility function weights) and how they impact policies	DTL
IN6	Cost/utility prediction	Expected costs/utilities depending on how utility function weights are chosen	DTL

quality attributes). To better understand the impact on policy actions, a categorization of policies into high-level collections (2) is required, so that humans do not need to reason directly about policy actions but can analyze more general strategies. An additional aspect is the role of utility function weights (3), whose impact is often obscure to stakeholders and needs to be clarified to help humans to correctly indicate their quality preferences (Wohlrab and Garlan, 2021b). Finally, given that we have a variety of variable types (4), support to analyze both quantitative and categorical data is required. Quantitative variables represent data as numerical values. Categorical variables assign one (qualitative) value of a fixed set of possible values to each policy (Starnes et al., 2010) and express properties in terms of values of enumerations, rather than as discrete or continuous numerical values.

To be useful in practice, the problem space of the planning context should be sufficiently large. Based on our experiences with the systems in this paper, the number of samples should be at least 200 and the number of variables no less than 50. These thresholds are easily met by most automated systems today.

As part of our approach, we apply a combination of Principal Component Analysis (PCA), Multiple Correspondence Analysis (MCA), clustering, and decision tree learning (DTL). Details about the employed machine learning techniques will be presented in Section 5.

When applying machine learning methods, dimensionality reduction techniques are commonly applied in combination with other techniques. For instance, combining dimensionality reduction and k-means clustering techniques can help to explore correlations and patterns in a dataframe (Markos et al., 2019). However, to the best of our knowledge, this combination of methods has not been applied to explain quality tradeoffs in automated planning.

Table 2 shows an overview of the information needs that are addressed by our approach and the specific techniques we recommend. To address requirement (1), we support general information needs (to explore variables, their contributions, and correlations) using PCA and MCA. PCA helps to focus on tradeoffs between quantitative variables, whereas MCA is useful to explore categorical variables and their relations. An example of a quantitative variable is the expected number of collisions that occur in a robot mission. An example of a categorical variable is the encoding of a decision that can be taken in a specific state (e.g., to move to another location or change the robot's speed). Using these dimensionality reduction techniques, the requirement to support quantitative and categorical data analysis is addressed (requirement 4). In particular, MCA can help users to detect actions impacted by a change to utility function weights. Moreover, using MCA, stakeholders can investigate which decisions are typically taken together and how they are connected to utility function weights and costs. Addressing requirement (2), we apply joint k-means clustering and PCA/MCA to identify clusters of policies (or strategies) that share similar characteristics, as well as their key attributes (e.g., the most frequent/infrequent values for categorical variables, percentages of policies that belong to a cluster, and the means of quantitative variables). To fulfill requirement (3), decision trees are useful to explore specific thresholds of utility function weights that distinguish different policies. They can also be leveraged to predict the expected costs/utilities (in total or related to specific quality attributes) depending on particular decisions or utility function weights.

4.2. Methodology and tool support

To apply our approach in practice, an implementation of a system needs to be in place, so that relevant data can be collected. The implementation can be at a prototype level or a final system. The crucial point is that it should be possible to specify a planning problem and execute a policy synthesizer (e.g., PRISM model checker Kwiatkowska et al., 2011). The planning problem is specified in the form of a Markov decision process (MDP), encoded in a plain text file. The MDP specifies one reward/cost structure per quality attribute and also a cost structure for the total cost, factoring in the different costs/rewards of the three quality attributes as a weighted sum using utility function weights. The MDP is used to generate policies that can be saved in a tabular format (e.g., using a csv file). In our examples, we performed uniform sampling of different combinations of utility function weights, but other sampling strategies are applicable as well. Instead of utility functions, for instance, priority values for non-functional requirements (Samin et al., 2022) can be used to capture the priorities of different quality attributes. The dataframe should have a sufficiently large number of samples to apply the machine learning algorithms (i.e., no less than 200). Once a dataframe file has been created, it can be input into appropriate statistics tools (e.g., R R. Core Team, 2020) and analyzed to generate explanations in the form of graphical plots.

The graphical plots can be created by following a sequence of steps. Fig. 2 shows an overview of our approach, which receives as input:

I1. a *planning problem specification* described by domain experts (e.g., modeled as a Markov decision process — MDP), and
I2. a set of *utility functions* and *weights* (or priorities) specified by various stakeholders.

I1. can be expressed as a plain text file containing a high-level model of an MDP (e.g., that can be processed by PRISM Kwiatkowska et al., 2011) and I2. can be expressed as a list of utility function weight combinations. Each utility function weight combination from I2. is used to generate an individual MDP, using the utility function weights when generating the total reward/cost structures.

This adjustment is performed in a Java program. The code and scripts can be found in our Github repository.[2] As mentioned

[2] https://github.com/cmu-able/planningTradeoffExplanations

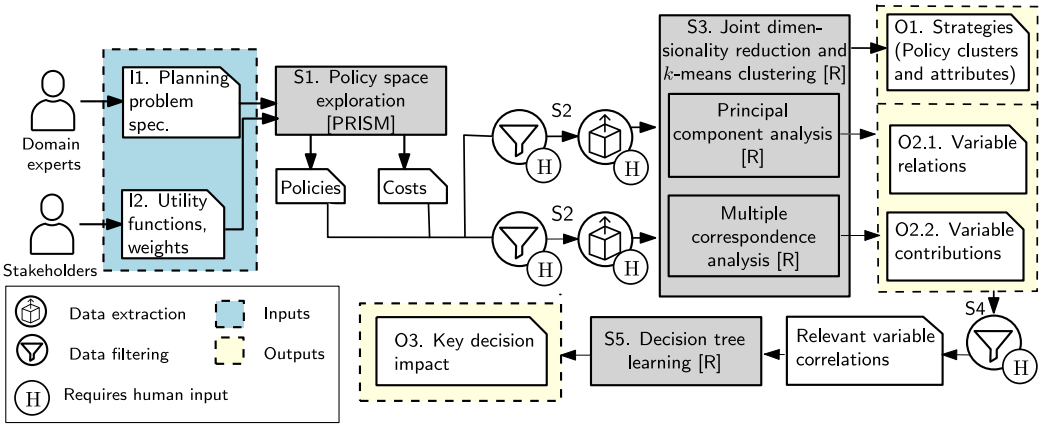


Fig. 2. Overview of our approach.

before, for the examples in this paper, we used uniform sampling and iterated over combinations of utility function weights with two loops. After creating an MDP, it uses the PRISM API to execute the model checker.

To facilitate understanding the solution space and how stakeholder preferences affect the outcome of the planning process, our approach provides the following outputs:

- O1. a set of *strategies* (*policy clusters and attributes*) that enables stakeholders to understand the various coarse-grained kinds of adaptation behavior that exist within the solution space (e.g., collision-avoidant vs. intrusiveness-avoidant), rather than requiring stakeholders to reason about numerous individual policies,
- O2. a set of *variable relations* (e.g., a strong preference for reducing travel time results in a lower number of collisions), as well as of *variable contributions* to quality variation in the solution space, and
- O3. a set of *key decision impacts* that helps to explain how the selection of specific values for key variables results in differences in the generated policies.

The approach consists of the following steps (cf. Fig. 2):

- S1. *Policy space exploration*, which consists of generating a set of policies that optimize different selections of utility preferences captured by varying weights. These policies (and their costs along the different quality dimensions) can be generated using an automated planner (e.g., PRISM Kwiatkowska et al., 2011).
- S2. *Data filtering and extraction* are employed for selecting candidate variables and extracting their values, preparing the data for subsequent processing steps. This step may require the input of a domain expert, who has to curate the set of candidate variables and extraction mechanisms, which can be reused across the same class of planning problem. This step produces as output two sets of dataframes that are used as input to PCA and MCA, respectively. For PCA, dataframes contain exclusively quantitative variables, whereas those for MCA contain mainly categorical variables, which can sometimes be complemented by quantitative variables.
- S3. *Joint dimensionality reduction and k-means clustering* employs both PCA and MCA to find variables that contribute in a meaningful way to QoS variation across the solution space, as well as their relations (i.e., the extent to which they are positively/negatively correlated). Using the results provided by PCA and MCA, this step applies a reduction and k-means clustering technique (Markos et al., 2019) that results in the elicitation of *policy clusters* (i.e., collections of policies that share similar characteristics), along with data on what the optimal number of clusters is and what key attributes distinguish a cluster from others.
- S4. *Data filtering* based on variable relations selects relevant variable correlations (i.e., those that are meaningful in terms of contribution to QoS variation), which can be employed as input to the next step. This step requires the input of a domain expert, whose decisions are informed by the output artifacts generated in the previous step.
- S5. *Decision tree learning* is used to arrive at the key decision variables that help to explain how the selection of concrete values for variables results in differences in the generated policies.

In our implementation, S1. is performed using PRISM (Kwiatkowska et al., 2011) and all subsequent steps are conducted using R (R. Core Team, 2020). For S2., humans might need to adjust the dataframes and select which variables are categorical/numerical, as well as which variables should be used for decision tree learning. The outputs of these steps are PDF files containing plots. These plots can be leveraged by humans to understand tradeoffs. The examples in this paper and the provided explanations indicate how to read these plots. In the future, we plan to provide further support for non-expert users (see Section 9).

While we use PRISM (Kwiatkowska et al., 2011) and R (R. Core Team, 2020) for the implementation, the approach we present in this paper can be supported with a variety of tools, depending on the concrete domain and system. In Section 8, we discuss the required steps to apply our approach to the exemplars for self-adaptive systems curated by the Software Engineering for Adaptive and Self-Managing Systems (SEAMS) community. In our implementation, we rely on R (R. Core Team, 2020), since it is free, open source, and available on a variety of platforms. To facilitate replication, we provide example dataframes and R scripts that can be used to generate the plots presented in this paper.[3]

4.3. Selected machine learning techniques

This section introduces the machine learning methods that are part of our approach: PCA, MCA, k-means clustering, and decision tree learning.

[3] https://github.com/cmu-able/planningTradeoffExplanations

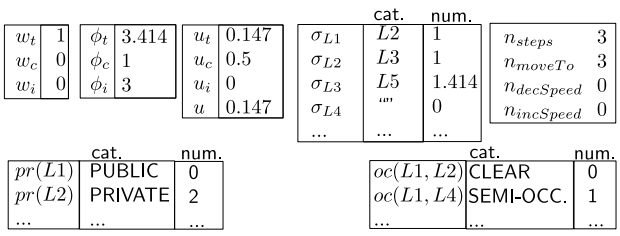


Fig. 3. Sample encoding of a policy for our robot mission planning example.

Principal Component Analysis (PCA). PCA (Jolliffe, 1986) is a statistical procedure used to reduce the dimensionality of datasets consisting of a large number of interrelated variables. The technique involves computing so-called *principal components* based on the data, which are new variables that indicate the directions in which the samples described by the variables differ. A principal component is a linear combination of the original variables that explains some of the variance in the data. The first principal component (PC1) carries the most information regarding the differences between samples in the data, the second principal component explains the most variance not covered by the first one, and so on. It is often the case that only the first two principal components carry relevant information (Jolliffe, 1986). The output of PCA includes (i) the percentage of total *variance* of the dataset explained by each principal component, (ii) correlation *loadings*, which describe how much variables contribute to explained variance, as well as their relationships, and (iii) *scores*, describing properties of the samples. In this paper, we are mainly concerned with variances and loadings, focusing on how much information is explained by the principal components and how variables are related to each other.

In the high-dimensional planning space for self-adaptive systems, PCA can be leveraged to visualize correlations between variables (e.g., utility function weights) and indicate how variables contribute to the differences in samples (i.e., generated policies).

Multiple Correspondence Analysis (MCA). Similar to PCA, MCA is a dimensionality reduction technique (Le Roux and Rouanet, 2009). MCA is applicable for categorical data. The decisions in our policies (e.g., that the robot should move to location L2 after visiting location L1) are encoded in categorical variables describing state–action pairs, which makes MCA an appropriate technique for analyzing policy data. While the focus is on categorical data, it is also possible to add supplementary quantitative variables when performing MCA. Variables and samples can be mapped into coordinates on principal axes. These axes are similar to principal components in PCA and help to explain the variance in the data, with the first axis being the most important to explain variance, the second axis being orthogonal to it and accounting for the second-most amount of explained variance, and so on. The output of MCA includes (i) the percentage of *variances* explained by each of the dimensions (axes), (ii) *coordinates and contributions of categories*, which describe how much each categorical variable contributes to the dimensions, as well as their relations, and (iii) *coordinates and contributions of samples*, describing properties of the samples.

Clustering Algorithms. Clustering is concerned with making sense of data by categorizing data points into collections that share similarities. k-means clustering (an unsupervised learning technique) is a widely used clustering technique and involves discerning k clusters in which every data point is allocated to the cluster with the nearest mean. We apply k-means clustering in combination with PCA and MCA (van de Velden et al., 2019; Markos et al., 2019) due to its potential of retaining as much variance as possible in as few dimensions as possible, while calculating and describing clusters of policies in the data. Reasoning about policies in terms of clusters is helpful for humans to understand patterns and characteristics of different categories of policies. The technique for joint dimensionality reduction and clustering requires the selection of parameters k (i.e., the number of clusters) and the number of dimensions n to consider for PCA and MCA. To select k and n, it is possible to calculate for what parameter values the clusters are most compact and the structure of clusters most well-separated using the so-called average silhouette width index (Markos et al., 2019). In our approach, we calculated the average silhouette width index for different cluster sizes to select k. There exist multiple evaluation techniques for clustering algorithms (Pfitzner et al., 2009). We manually labeled the samples in our dataset to calculate the precision, recall, and F1 scores of the algorithms. To evaluate the distance between clusters, we use the average silhouette width index, since it indicates how similar a policy is to other policies in its cluster and how different it is to other clusters (Batool and Hennig, 2021).

Decision Tree Learning. Decision tree learning is a supervised learning technique that predicts the value of a variable based on the values of other variables (Breiman et al., 2017). In our case, we apply decision tree learning both to grow classification trees (to be able to predict what decision is taken when the system is in a certain state) and regression trees (e.g., to predict costs in different quality dimensions, depending on actions taken by the system).

Summary. To summarize, we have outlined the employed machine learning techniques of our approach: PCA, MCA, clustering, and decision tree learning. PCA (Jolliffe, 1986) and MCA (Le Roux and Rouanet, 2009) are dimensionality reduction techniques used to focus the analysis on key components/dimensions that describe the differentiating features in the data. These dimensionality reduction techniques are combined with clustering algorithms (van de Velden et al., 2019; Markos et al., 2019) to identify strategies (collections of policies with similar characteristics). To perform a fine-grained analysis of those strategies, decision tree learning (Breiman et al., 2017) is used to indicate variable thresholds (e.g., in utility function weights) and their impact on policies.

4.4. Data filtering and extraction

Our approach requires human input for data filtering and extraction. Concretely, human input is required to select relevant variables for a specific system and how they should be represented in dataframes. The data is gathered in two types of dataframe,[4] one containing a mix of categorical and numerical variables (for MCA) and the other one containing numerical variables only (for PCA). Fig. 3 shows an excerpt of the extracted data of a policy for our robot mission planning example. In the following, we leverage this example to explain how policy data is encoded. To extract the data that corresponds to the columns that characterize a policy in a dataframe (MCA or PCA), we define the notion of *policy encoding*, which is just a specification of the action selections made by the policy at each of the decision points (i.e., nondeterministic choices) in the MDP:

Definition 4.1 (*Policy Encoding*)**.** The encoding of a policy $\sigma : S \to A$, denoted henceforth by $e : S \times A \to S \times A$, is a projection of the policy that only enumerates state–action pairs in which action selection has been performed in the original MDP $\mathcal{M} = (S, s_I, A, \Delta, L, R)$, i.e., $e(\sigma) = \{(s, a) : \sigma \mid |A(s) > 1|\}$.

[4] A dataframe is a two-dimensional data structure capturing samples (in rows) with their characteristics (in columns of potentially different types).

The tables in the top left in Fig. 3 represent the utility function weights $w_* \in x_W$, the costs for different quality attributes $\phi_* \in x_C$, and the utilities $u_* \in u_\sigma$ (as introduced in Section 2). The policy in Fig. 3 is encoded in $\sigma_* \in x_E$, where σ_s describes the action to be taken in a state s. In our example, we consider a fixed speed of 1 and the state is defined by a location on the map. In the context of other robot planning scenarios, the state might be defined, e.g., by a pair describing the location and the robot's speed.

Once we have defined the encoding of a policy, we build upon it to describe our *categorical dataframe* for MCA. The categorical dataframe contains categorical variables that assign one (qualitative) value of a fixed set of possible values to each policy (Starnes et al., 2010). In particular, the actions taken by a policy are encoded as categorical variables (e.g., with the qualitative value "moveTo(L7)"). The categorical dataframe also contains quantitative variables, e.g., for the policies' utility function weights and costs.

Definition 4.2 (*Categorical Dataframe*)**.** For an output tuple $\lambda = (\sigma, u_\sigma, u_\sigma^g)$ generated by the planner from the policy exploration profile $\tau = (\mathcal{M}, W, C_\Phi, U, \Pi)$, a row for a categorical dataframe consists of the concatenation of the elements of the tuple $(x_W, x_C, u_\sigma, x_E, x_P)$, where:

- $x_W \in \tau.W$ is the set of weights from which λ was generated.
- $x_C \in \mathbb{R}^n_{\geq 0}$ is the set of quantified costs of σ along the different quality dimensions.
- $x_E = e(\sigma)$ is the policy encoding of σ.
- $x_P \in \tau.\Pi$ is the set of parameter values from which λ was generated.

A categorical dataframe consists of the aggregation of all the rows resulting from encoding the tuples contained in policy exploration profile τ.

For the categorical dataframe (cat. in Fig. 3), actions indicate the location that the robot should move to or the speed that should be set.

In addition to categorical dataframes, we have *numerical dataframes* for PCA, which are obtained in a similar way, but with additional constraints that impose numerical types for all the values contained in the rows. Numerical dataframes do not contain any categorical variables, but convert them into continuous or discrete numerical values. The policy actions are transformed into real values (e.g., indicating the traveled distance), as we describe below.

Definition 4.3 (*Numerical Dataframe*)**.** For an output tuple $\lambda = (\sigma, u_\sigma, u_\sigma^g)$ generated by the planner from the policy exploration profile $\tau = (\mathcal{M}, W, C_\Phi, U, \Pi)$, a row for a numerical dataframe consists of the concatenation of the elements of the tuple $(x_W, x_C, u_\sigma, x_E, x_P)$, where x_W, x_C and u_σ are defined in the same way as for the categorical dataframe, with the additional constraint that all parameters in x_P belong to numerical types, and the elements in x_E are transformed via a function $f_{e\sigma} : S \times A \rightarrow S \times \mathbb{R}$ that maps policy actions to numerical values.

For the numerical dataframe (num. in Fig. 3), actions are translated into numerical values and represent the distance that the robot should travel in this state (and 0 if it does not visit the location), i.e., $f_{e\sigma}(s, a) = (s, d(s, s'))$, where $d(s, s')$ is the distance between states (locations). Apart from the policy encoding in state–action pairs, variables are used to describe the number of steps n_{steps} in the policy (i.e., the number of actions that are taken from the start to the goal location), the number of movements n_{moveTo}, and the number of steps in which the speed is decreased or increased ($n_{decSpeed}$ and $n_{incSpeed}$).

The tables at the bottom of Fig. 3 show environmental parameters (in x_P, cf. Definitions 4.2 and 4.3) that in this case represent characteristics of the map, i.e., the privacy levels of the locations L*i* and the occlusion levels of all edges (L*j*, L*k*). These variables are represented as categorical variables in the dataframe for MCA (e.g., PUBLIC, PRIVATE, CLEAR, …) and as numerical variables in the dataframe for PCA. With the numerical variables, we indicate whether a node is public (0), semi-private (1), or private (2), and whether a path between two nodes is clear (0), partially occluded (1), or occluded (2). Moreover, characteristics of the map are captured in order to analyze how variations in the map result in different generated policies. These variables are aggregated with the policy data to form a single dataframe for analysis. While the examples shown in this paper do not incorporate any map variations, it is useful to consider environmental variables due to their substantial influence on the adaptation behavior generated by an automated planner.

In our dataframes, to focus the subsequent analysis on differentiating variables, variables that have the same value for all rows are removed. Moreover, as part of PCA, the data is scaled to have unit variance before the analysis takes place.

When applying the approach to real-world systems, human input is needed to select which variables are relevant in the particular context and how they should be represented and extracted. For example, methods can be added to existing planning programs that extract the required data as csv files. Human input is also required to select which variables should be considered when creating decision trees. Details about the filtering step before DTL will be provided in Section 5.4.

5. Analyzing quality tradeoffs using machine learning techniques

This section describes how machine learning methods are applied as part of the approach: PCA, MCA, k-means clustering, and decision tree learning.

To illustrate the techniques, we use a robot mission planning example that is based on the system introduced in Section 2. The extended version relies on a map containing 69 nodes (39 public, 14 semi-private, and 16 private) and 194 edges (15 of which are partially occluded and 16 occluded). The privacy and occlusion properties were randomly assigned to nodes and edges. This random assignment of properties can contribute to the results of the quality tradeoff analysis — if the most commonly selected paths happen to be public, privacy might be less significant for the tradeoff among quality attributes.

5.1. Principal Component Analysis (PCA)

We apply PCA to visualize correlations between variables (e.g., utility function weights) and contributions of variables to the differences in policies. Fig. 4 shows a PCA *correlation loading plot* for our robot mission planning example. Table 3 indicates the variables used in the plot with their meanings.

The plot indicates which variables are most relevant to explain the variance in the dataset and their relations to the first two principal components. The first principal component (PC1) accounts for 44.4% and the second (PC2) for around 30% of the explained variance in the data. The plot contains two ellipses indicating how much variance is taken into account, with the outer ellipse being the unit circle and accounting for 100% explained variance and the inner ellipse indicating 50% explained variance. Variables between the edges of the two ellipses are the most distinguishing variables and the closer to the unit circle, the more important a variable is. It can be seen that the utility function weights contribute to different extents: `w_intrusiveness`

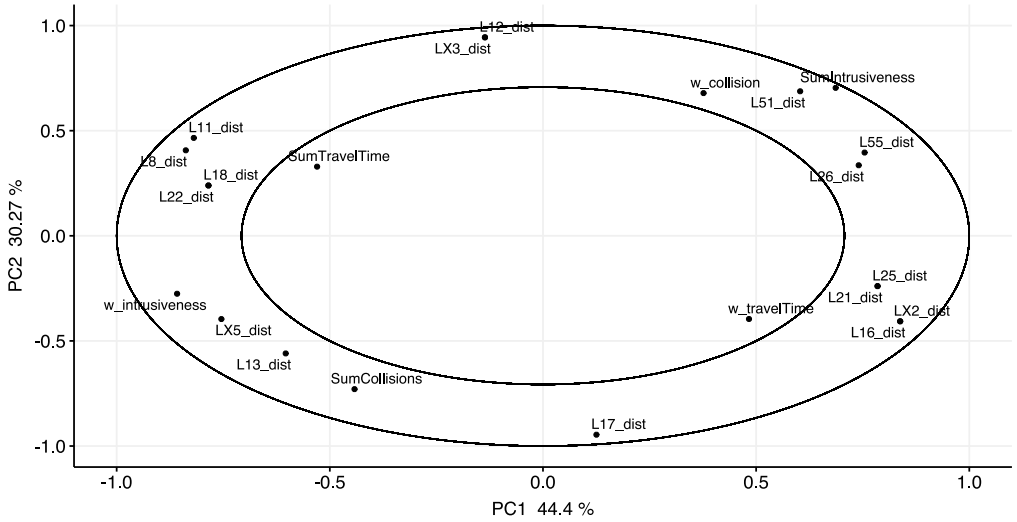


Fig. 4. PCA correlation loading plot for our robot mission planning scenario.

Table 3
Variables shown in the PCA correlation loading plot (Fig. 4).

Variable	Meaning
L8_dist	Distance traveled starting from location L8
SumCollisions	Sum of expected collisions (with occluded segments counting 2 and semi-occluded segments counting 1)
SumIntrusiveness	Sum of intrusive incidents (with visited private locations counting 2 and semi-private locations counting 1)
SumTravelTime	Sum of travel time
w_intrusiveness	The importance of not missing a target (between 0 and 1)
w_travelTime	The importance of not being destroyed (between 0 and 1)
w_collision	The importance of not being destroyed (between 0 and 1)

is of high relevance for PC1 and is negatively correlated with `w_travelTime` and `w_collision`. Overall, `w_intrusiveness` has the greatest impact on the variance in the data, being located between the unit circle and the circle explaining 50% of the variance. It is followed by `w_travelTime` as the second most differentiating utility function weight. The weight of collision is of less importance, since it mainly impacts PC2.

Besides analyzing the contributions of variables using the plot, it is also possible to see the relationships between them. The angle between the vectors going from the origin of coordinates to two variable points indicates how closely these variables are correlated. Fig. 4 indicates a positive correlation between `w_intrusiveness`, `SumCollisions`, and `SumTravelTime` (all of which are negatively correlated with `SumIntrusiveness`). This observation indicates that if intrusiveness is an important attribute, the sums of collisions and travel time will be high and the sum of traversed intrusive locations will be low. Moreover, variables whose points are on opposite sides of the plot and form an angle larger than 90 degrees (e.g., there is an angle close to 180 degrees between the vectors going from the origin of coordinates to the variable points) are negatively correlated, e.g., `w_travelTime` and `SumTravelTime`. Finally, it can be seen that `w_collision` is positively correlated with `SumIntrusiveness` (which are both negatively correlated with `SumCollisions`), indicating that collisions are often avoided by accepting a high sum of traversed intrusive locations. These observations are in line with what one would expect of this robot mission planning example: policies either optimize for intrusiveness avoidance (at the expense of travel time and collision avoidance), travel time (at the expense of collision avoidance) or collision avoidance (at the expense of intrusiveness). Analyzing these correlations facilitates the understanding of tradeoffs in the quality space. Another conclusion illustrated by the plot is that all utility function weights are negatively correlated with each other. Hence, stakeholders need to decide which quality attribute(s) to prioritize, which will entail differences in the cost of the observed policies (expressed in the sums of travel time, collisions, and traversed intrusive locations).

Table 4
Variables shown in the MCA biplot (Fig. 5).

Variable	Meaning
LX3_	Decision indicating that LX3 is not visited
LX3_moveTo(L18)	Decision indicating that the action in LX3 is to move to L18

5.2. Multiple Correspondence Analysis (MCA)

MCA is similar to PCA in the sense that it helps to reduce the dimensionality of a dataframe and indicates correlations between variables. The output of MCA is often shown as a two-dimensional biplot in which individual samples and variables are shown along the first and second principal axes. For the robot mission example, an MCA biplot is shown in Fig. 5. The variables with their meanings are clarified in Table 4. The locations of the samples are plotted as blue points and categorical variables are indicated as red triangles. The variables are plotted with respect to the two principal dimensions. The two first dimensions account for 46% and 31.8% of the explained variance. The squared correlations between variables and the dimensions are used as coordinates. Variables that are close to each other are correlated with each other, for instance, `L12_moveTo(LX3)` and `LX3_moveTo(L18)`. It can be seen that several groups of categorical variables (red triangles) are strongly correlated with each other and with the depicted samples (blue points indicating policies). Variables that

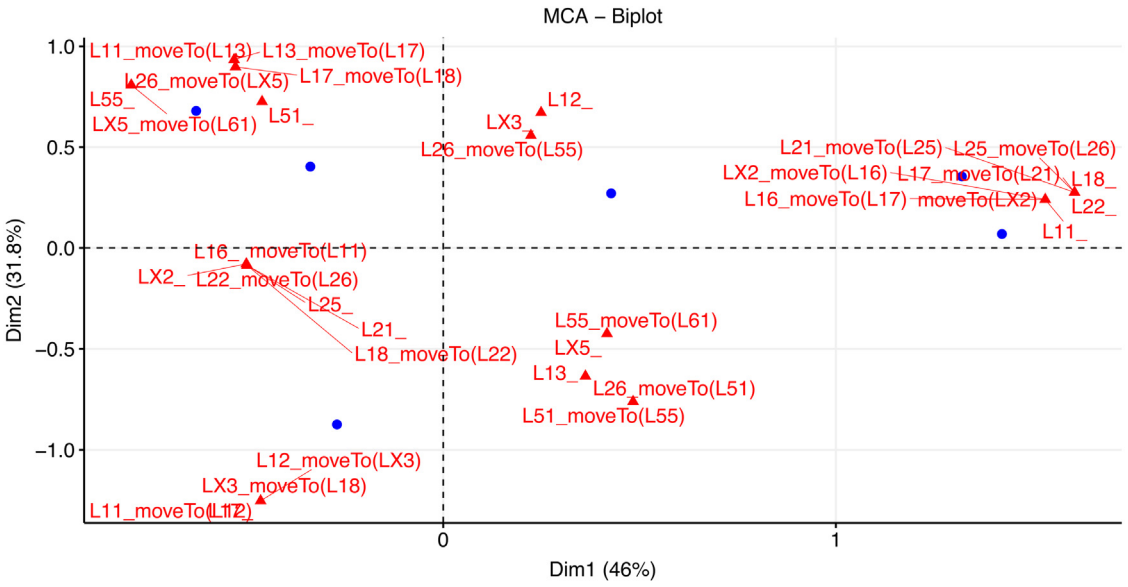


Fig. 5. MCA biplot for our robot mission planning scenario. Samples are plotted as blue points and categorical variables are shown as red triangles.

are close to 1 and −1 on an axis contribute strongly to that dimension. For instance, the cluster of categorical variables in the bottom left of the plot indicates that actions to move the robot from L11 via L12 and LX3 to L18 are commonly taken together and are correlated with a group of policies. Joint dimensionality reduction and clustering allow users to explore this observation further.

Already at first glance, the MCA biplot indicates that there might exist different groups of samples and variables (which can be considered as corresponding to different classes of policies). Our approach supports further investigation of these groups of policies with the joint dimensionality reduction and clustering method described below.

5.3. Clustering algorithms

To select an appropriate k for the number of clusters, we calculated the average silhouette width index for different cluster sizes. The average silhouette width index was highest for $k = 4$ and 3 dimensions, indicating that for these values, the clusters are most compact and well-separated. In this example, the average silhouette width index was 0.9604. Average silhouette width index values are between −1 and +1, with values close to 1 indicating the clusters are well-separated. A value of 0.9604 indicates that different clusters are well-distinguished from each other (Lengyel and Botta-Dukát, 2019). Therefore, we chose $k = 4$ and $n = 3$ for our subsequent analysis.

Besides calculating the average silhouette width index, we also calculated macro-averaged precision, recall, and F1 scores. The precision was 0.916, the recall 0.850, and the F1 score 0.873. These values indicate that the automatically calculated clusters correspond well to the clustering based on manual labeling.

In Fig. 6, a reduced k-means biplot for the robot mission planning example is shown, in which policies are indicated as points and quantitative variables (e.g., utility function weights and policy costs) are shown as axes. As a first step, we combined PCA with k-means for clustering (Markos et al., 2019). Policies are grouped into four clusters that are correlated with quantitative variables to different degrees. The clusters are labeled as FC (fast cluster), BC (balanced cluster), CAC (collision-avoidant cluster), and IAC (intrusiveness-avoidant cluster). Note that these labels were manually created and are not automatically assigned to the clusters. As in the PCA correlation loading plot, the angle between the vectors going from the origin of coordinates to two points indicates how closely these variables or samples are correlated. The closer the point representing a policy is to the label of an axis, the stronger is the correlation. The data suggests that there is a substantial difference between the clusters: CAC (the collision-avoidant cluster) is correlated with a high weight of collision, whereas IAC (the intrusiveness-avoidant cluster) is correlated with a high weight of intrusiveness. FC (the fast cluster) is mainly correlated with a high weight of travel time and BC (the balanced cluster) is correlated (to a lesser degree) with `w_travelTime` and `w_collision`. Moreover, CAC is associated with a high sum of traversed intrusive locations, IAC with a high sum of collisions, and FC and BC with a short travel time. It should also be noted that `L8_dist`, `L11_dist`, `L17_dist`, and `L26_dist` are policy variables shown in the plot, which indicates that the decisions at those locations are the most important to characterize differences between clusters. Going back to Fig. 5, we can see that the decisions taken at these locations are indeed distinguishing characteristics of different groups of policies. For large-scale examples such as this one, it is often not obvious for humans to see what decisions differentiate clusters of policies, which is why these plots are useful.

Fig. 7 depicts how the clusters can be further characterized. The cluster means for the quantitative variables in all clusters are shown. Similarly to our previous observations, the figure indicates that CAC (the collision-avoidant cluster) is correlated with a high weight of collision, a high sum of traversed intrusive locations, and a low sum of collisions. IAC (the intrusiveness-avoidant cluster) is correlated with a high weight of intrusiveness, a high sum of collisions, and a low sum of intrusive traversed locations. FC (the fast cluster) is correlated with a high weight of travel time, low sums of collisions and travel time, and a high number of traversed intrusive locations. Finally, BC (the balanced cluster) deviates less from the mean of 0 of the standardized values across all clusters. BC has a slightly increased mean weight of travel time, which reflects in low sums of travel time and intrusiveness and a slightly higher sum of collisions.

Apart from the coordinate plots, it is also worth examining how the clusters can be characterized in terms of categorical variables, which is something that can be done by applying MCA in combination with k-means clustering (Markos et al., 2019).

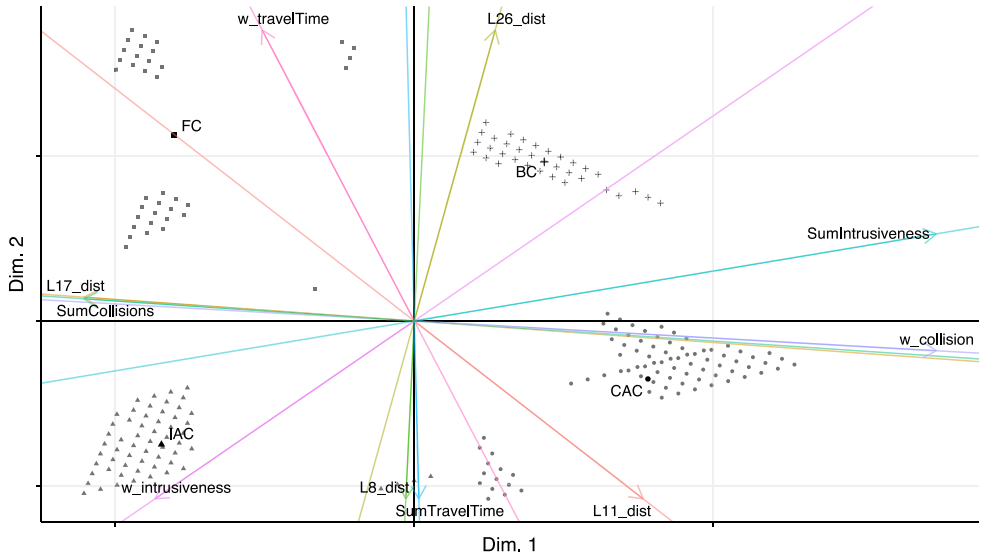


Fig. 6. A reduced k-means biplot of policies (points) and quantitative variables (axes with respect to Dimensions 1 and 2). The clusters are indicated as CAC, IAC, FC, and BC.

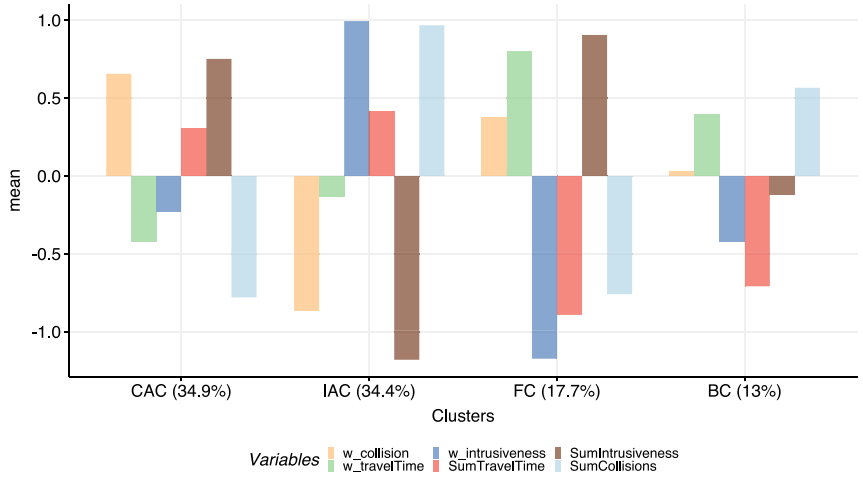


Fig. 7. Bar plot of the cluster means for our robot mission planning scenario.

Fig. 8 shows the values of categorical variables that have the highest standardized residuals (both positive and negative ones) for each cluster. Standardized residuals represent how much the values of a categorical variable among the samples in a cluster differ from the values of that variable across *all* samples, taking into account their standard deviation. Variables with a high absolute standardized residual are the ones that have an above/below average frequency in a cluster. In this example, the values encode state–action pairs, indicating the location at the current state, followed by the action that should be taken (or an empty value if the policy does not enter the state). For instance, `L11.moveTo(L12)` in Fig. 8 indicates that for CAC, an unusually frequent decision at location L11 is to move to L12, which has a high standardized residual. Policies in FCAC frequently include variables indicating that the robot shall move from L11 to L12, from L12 to LX3, and from LX3 to L18. For IAC, it can be seen that policies in that cluster commonly move from L11 to L13 and from L13 to L17 (given that those values have high standardized residuals). Policies in FC and BC frequently avoid visiting L11 at all. In the particular map taken for this example, L11 and L12 are private nodes, whereas L13 is a public node. This observation indicates why policies in the intrusiveness-avoidant cluster IAC tend to move to a public, non-intrusive location (i.e., L13) in this state.

Based on the analysis of standardized residuals, it is possible to elicit which decisions (in terms of state–action pairs) differentiate clusters. In order to analyze how changes in the utility function would lead to different policies being generated, we focus especially on these differentiating decisions when applying decision-tree learning.

5.4. Decision tree learning

We applied decision tree learning to dive into the details and explain how, for example, a different selection of utility function weights would impact the generated policies.

Before applying decision tree learning as part of our approach, a filtering step needs to be performed in which relevant variables are identified. Decision tree learning requires as an input a dependent variable Y whose value should be predicted based on a number of independent variables X. In case a classification tree is grown, Y is a categorical variable, whereas for regression trees, Y is quantitative. In our approach, various combinations of variables are of interest, for instance:

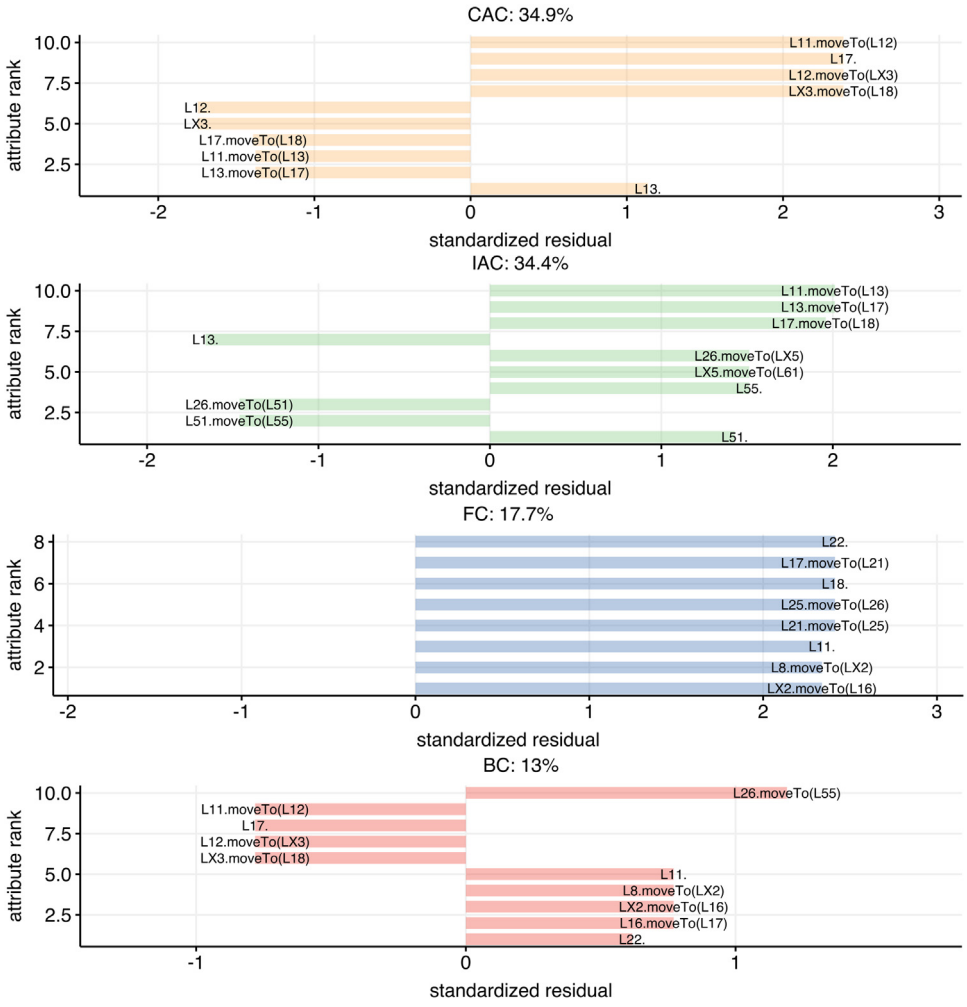


Fig. 8. Variables with the highest standardized residuals (positive or negative) for each cluster in the robot mission planning scenario.

1. predicting what action is taken in a state s, depending on the utility weights w, i.e., $Y = \sigma(s)$ and $X = w$ (a classification tree);
2. predicting what action is taken in state s_j, depending on the actions that were taken in previous states, i.e., $Y = \sigma(s_j)$ and $X = \{\sigma(s_1), \ldots, \sigma(s_{j-1})\}$ (a classification tree);
3. predicting the utility obtained in the quality attribute dimension i, depending on the selected policy σ, i.e., $Y = u_{\sigma i}$ and $X = \sigma$ (a regression tree);
4. predicting the cost in a quality attribute dimension i, depending on the selected policy σ, i.e., $Y = \Phi_{\sigma i}$ and $X = \sigma$ (a regression tree).

When predicting what action is taken in a state, we suggest focusing on variables that have high absolute standardized residuals in several clusters. L11 in our robot mission planning example is such a state in which the selected actions differ strongly in different clusters and depending on the chosen utility function weights.

Fig. 9 shows an example of a decision tree plot for the robot mission example. It depicts a classification tree that indicates what actions shall be taken when the robot is at location L11. It indicates what utility function weights the decision depends on. Decision trees can be read in a top-down manner. Generally, for low collision and high intrusiveness weights, the decision is to move to L13, whereas for low intrusiveness weights and high weights of travel time, the decision is to avoid L11. Moving to L12 is the decision taken for in-between values, e.g., for a weight of 0.3 for intrusiveness, a weight of 0.3 for collision avoidance, and a weight of 0.4 for travel time. The decision tree plot can support stakeholders in further analysis of the spectrum of utility function weights and their impact on selected policy actions. It should be noted that the condition at the root of the tree is connected to the most differentiating variable in the dataframe. In the example above, it is the weight of intrusiveness. While this observation confirms our findings from PCA, it is not sufficient to solely rely on decision tree learning for quality tradeoff explanations, as tradeoffs are not as evident in decision trees.

5.5. Summary

To summarize, with PCA (Jolliffe, 1986) and MCA (Le Roux and Rouanet, 2009), our approach helps to reduce the dimensionality of our data and identify the differentiating variables. As a next step, the combination of dimensionality reduction techniques with clustering (van de Velden et al., 2019; Markos et al., 2019) is used to identify strategies (categories of policies with similar characteristics). To analyze the details of these strategies and identify thresholds of variable values, decision tree learning (Breiman et al., 2017) is employed.

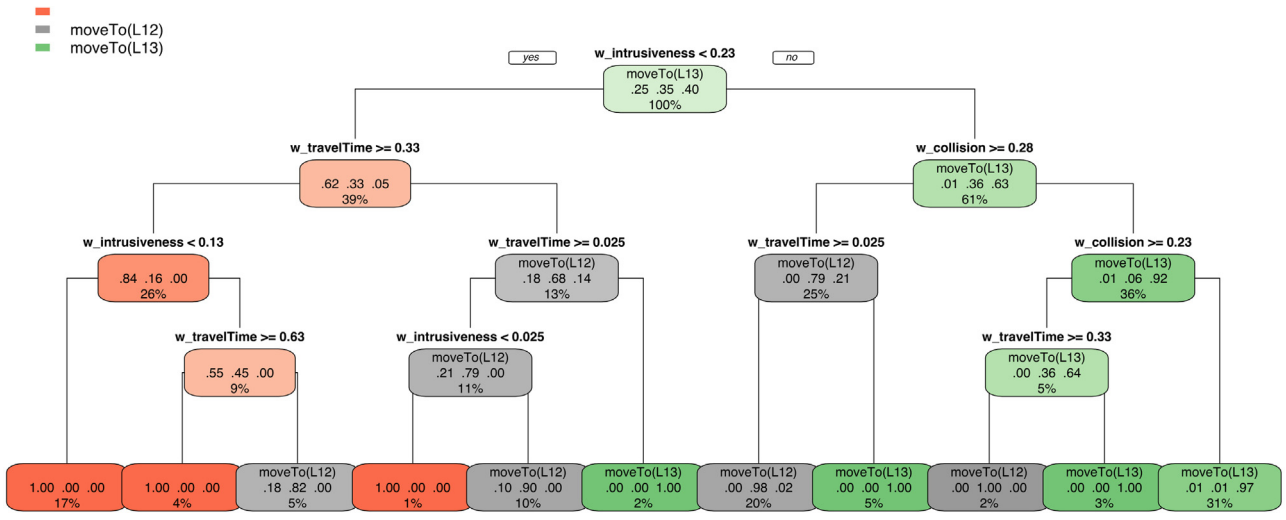


Fig. 9. Decision tree plot for the robot mission planning example (showing the selected decision in the state L11).

6. Evaluation

In the following, we present our evaluation method (Section 6.1), the results of our evaluation (Section 6.2), and threats to validity (Section 6.3).

6.1. Evaluation method

We evaluated our approach with respect to the feasibility of applying it to other automated planning problems, as well as with respect to the tradeoff between the information reduction and the amount of explained variance retained by the results obtained with our approach. Concretely, our research questions are:

1. **RQ1:** To what extent is the approach applicable to automated planning problems? (Feasibility)
2. **RQ2:** How much can the complexity of the information presented to a human stakeholder be reduced while preserving most of the relevant information? (Information reduction vs. amount of explained variance, only relevant to PCA and MCA)

The evaluation focuses on the approach's feasibility (RQ1) to investigate how applicable machine learning-based techniques are to elicit and explain tradeoffs. With feasibility, we refer to the extent to which it is possible to generate explanations for automated planning systems and draw tradeoff-related conclusions from them. Analyzing the information reduction-explained variance ratio (RQ2) gives indications about the potential of our approach to reduce the complexity of large problem spaces while retaining important information. Note that the focus does not lie on the usability of our approach, but on the applicability and potential of applying machine learning techniques to explain tradeoffs. In the future, the answers to these research questions can form a basis to develop more focused and usable explainability techniques (see Section 9).

We used two systems for evaluation: an extended version of the robot mission planning example (see Section 5) and DARTSim, concerned with a fleet of drones that perform a mission (Section 6.2.1). For both systems, we generated plans based on a fixed planning problem and a set of utility functions (using uniform distributions over the space of utility function weights) with the probabilistic model checker PRISM (Kwiatkowska et al., 2011).

6.2. Evaluation results

This section presents the evaluation results. Table 5 shows an overview of the dataframe dimensions we considered for the experimental evaluation described in this paper. The dimensions relate to the number of quality attributes, cost dimensions, the number of states, the number of action types, and the number of samples we collected using PRISM. The number of states is connected to the number of locations on the map (for robot mission planning) and with the number of combinations of segments, altitude, and configurations that the fleet of drones can be in (for DARTSim). The action types represent the actions to move to another location or adjust the speed to fast or slow (for robot mission planning) and to adjust the altitude, move, or changing the configuration (for DARTSim). In the following, we describe how the approach was applied to the DARTSim systems to assess its feasibility (RQ1, Section 6.2.1). We then describe the evaluation results of the information reduction vs. amount of explained variance tradeoff (RQ2, Section 6.2.2).

6.2.1. DARTSim system (RQ1)

The second example system used for evaluation was DARTSim, which is an exemplar that originated from the DART (Distributed Adaptive Real-Time) project (Moreno et al., 2019). The system is concerned with a fleet of drones that attempts to detect targets while avoiding being hit by a threat. We used a version of DARTSim in which electronic countermeasures (ECM) can be switched on or off and the fleet of drones can switch between loose and tight formations. Using ECM and flying in tight formations reduce the probability of being destroyed, but also decrease the probability of detecting a target. Moreover, the drones fly through five segments before arriving at their goal location and their altitude levels range from 1 to 4. The initial position is in segment 1 in a loose formation with ECM switched off and at an altitude of 1. The relevant quality attributes are safety (measured by the probability of being destroyed) and the success of the mission (measured by the probability of missing a target). Possible actions in each state are to fly to the next segment, increase/decrease the altitude by 1 or 2, wait, change the formation to loose or tight, and switch ECM on or off. To capture the policy actions in the numerical dataframe used for PCA, we converted the actions into numerical values capturing the change in altitude in a $[-2, +2]$ interval.

In Fig. 10, the PCA correlation loading plot for DARTSim is depicted, where PC1 accounts for 45.57% and PC2 for 19.73% of the explained variance. Table 6 shows an overview of the variables

Table 5
Dataframe dimensions for experimental evaluation.

System	QAs	Costs	# States	Action types	# Samples
Robot mission planning	3	3	69	196	215
DART	2	2	80	10	200

Table 6
Variables shown in the PCA correlation loading plot (Fig. 10).

Variable	Meaning
costDestroyed	Cost of the drone fleet being destroyed
missedTargets	Number of missed targets
NumberDecAlt	Number of steps in which the fleet decreases its altitude
NumberFly	Number of steps in which the fleet flies
NumberIncAlt	Number of steps in which the fleet increases its altitude
NumberSteps	Number of steps in total in the current mission
seg.1.form.tight.ecm.true.	Change of altitude (between −2 and 2) when the fleet is in segment 1 at
destr.false.alt.1	altitude 1, in a tight formation, with ECM switched on, and not destroyed
w_missTarget	The importance of not missing a target (between 0 and 1)
w_destrProbability	The importance of not being destroyed (between 0 and 1)

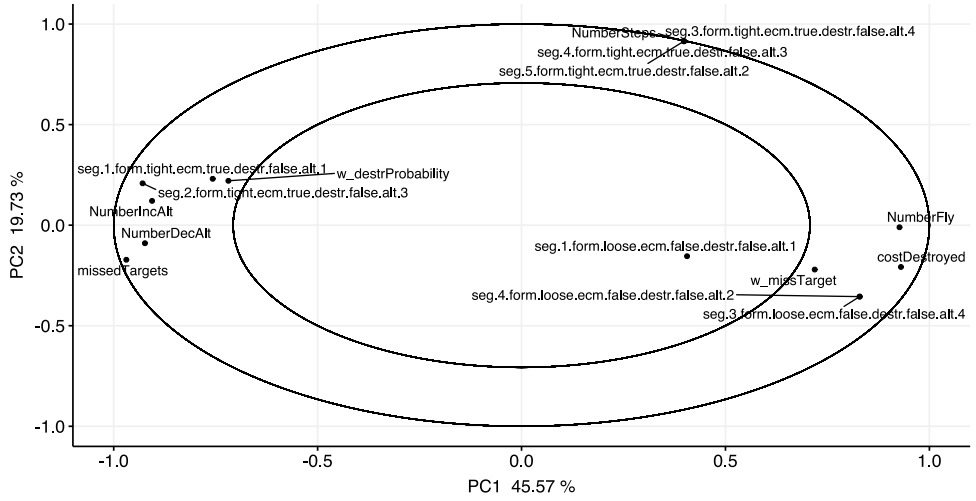


Fig. 10. PCA correlation loading plot for DARTSim (all variables).

and their meaning. It can be seen that there exist two negatively correlated clusters of quantitative variables: on the left side of the plot, `w_destrProbability` is located, along with the sum of missed targets, the altitude increase at segments 1 and 2 (in tight formations with ECM switched on), and the number of steps in which the altitude is increased and decreased. On the right side of the plot, `w_missTarget` is located, which is correlated with the cost of getting destroyed, the number of steps in which the drones fly, and altitude changes at segments 1, 3, and 4 (in loose formations with ECM switched off). We can see that the two utility function weights are negatively correlated with each other. The observations reflect what can be expected of this system: it either optimizes policies in order to avoid being destroyed (and therefore, it uses tight formations and ECM) or it optimizes for target detection, uses loose formations and no ECM, and gets destroyed more often. Among the categorical variables depicted here, `seg.2.form.tight.ecm.true.destr.false.alt.3` is the most relevant, as it is located close to the unit circle and central to explain PC1. We observed that the relevant categorical variables are often associated with the differentiating decisions in the policies and are good candidates to explore as one of the input variables for decision tree learning.

Fig. 11 shows an MCA plot of selected categorical variables and policies. The variables with their meanings are clarified in Table 7. Given that there exist many categorical variables in this example and plotting them would lead to overlapping labels, we filtered the data by the variable previously identified as relevant (i.e., those that have the initial state `seg.2.form.tight.ecm.true.destr.false.alt.3`). Hence, the plot depicts the categorical variable values for actions selected in segment 2 when the fleet is in a tight formation with ECM switched on. It can be seen that three main groups of policies exist that are correlated with the drones (i) decreasing their altitude by 2, (ii) decreasing their altitude by 1 or flying to the next segment, or (iii) increasing their altitude by 1.

When it comes to clustering, in this example the optimal cluster size differs when calculating it based on the MCA dataframe or based on the PCA dataframe. The resulting clusters are either five clusters of sizes 25.5%, 23.5%, 18%, 17%, and 16% (using the PCA dataframe) or three clusters of size 81.5%, 9.5%, and 9% (using the MCA dataframe). Given that the MCA dataframe reflects the main characteristics of policies more explicitly than the PCA dataframe (where we abstract from actions being taken by representing them using numerical values, i.e., altitude changes in the case of DARTSim), we decided to use $k = 3$ for our analysis.

A reduced k-means biplot for DARTSim, indicating three clusters of policies along with quantitative variables, is depicted in Fig. 12. The closer the point representing a policy is to the label of an axis of a quantitative variable, the stronger is the correlation. The data suggests that there is a substantial difference between the clusters: RTC (the risk-taker cluster) is correlated with a high weight of not missing a target, whereas DAC (the destruction-avoidant cluster) is correlated with a high weight of avoiding

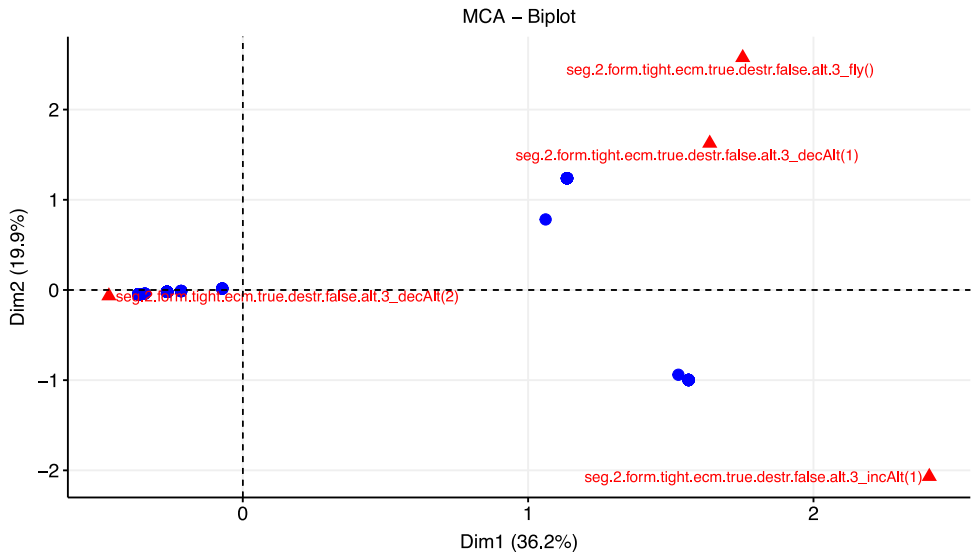


Fig. 11. MCA biplot for DARTSim. Samples are plotted as blue points and categorical variables are shown as red triangles.

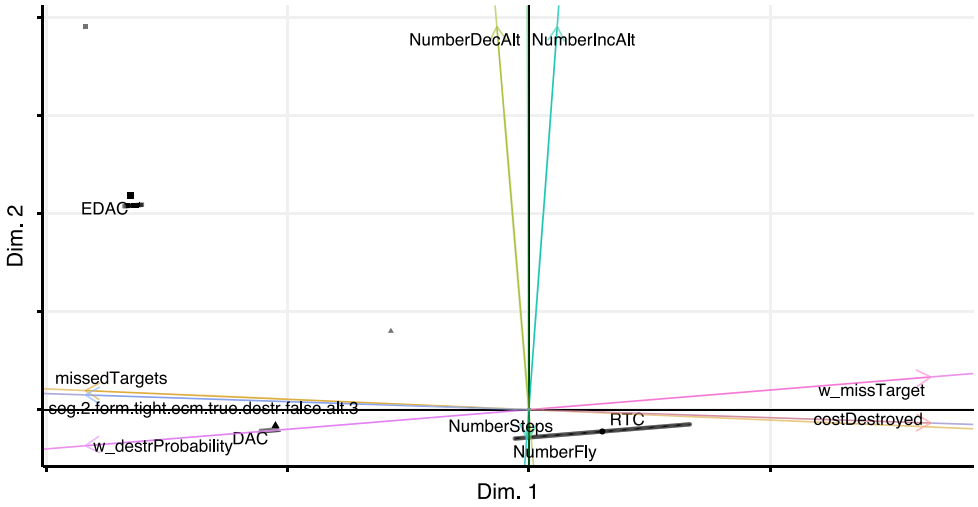


Fig. 12. A reduced k-means biplot of policies (points) and quantitative variables (axes with respect to Dimensions 1 and 2) of DARTSim. The clusters are indicated as RTC, DAC, and EDAC.

Table 7
Variables shown in the MCA biplot (Fig. 11).

Variable	Meaning
seg.2.form.tight.ecm.true. destr.false.alt.3_fly()	Decision indicating that in segment 2 with altitude 3, tight formation, ECM activated, and no destruction, the drone fleet should continue flying at the same altitude
seg.2.form.tight.ecm.true. destr.false.alt.3_decAlt(2)	Decision indicating that in segment 2 with altitude 3, tight formation, ECM activated, and no destruction, the drone fleet should decrease its altitude by 2

destruction. EDAC (the extremely destruction-avoidant cluster) is correlated with a high weight of avoiding destruction and tends to change its altitude more frequently (resulting in a correlation with `NumberDecAlt` and `NumberIncAlt`). Moreover, it can be seen that RTC is correlated with a high cost of being destroyed, whereas DAC and EDAC are correlated with a high number of missed targets. These observations confirm our previous findings regarding the tradeoff between detecting more targets vs. not being destroyed.

Fig. 13 gives further insights regarding the clusters' characteristics: RTC (the risk-taker cluster) has a slightly increased weight of not missing a target, a high cost of destruction, a low weight of avoiding being destroyed, and a low number of missed targets. DAC (the destruction-avoidant), on the other hand, has a high weight of avoiding being destroyed and a low weight of not missing a target, resulting in a high number of missed targets and a low cost of destruction. EDAC (the extremely destruction-avoidant cluster) has similar weights as DAC, however, its weight of avoiding being destroyed is even higher and the weight of not missing a target is even lower. The extreme preference is reflected in the low cost of being destroyed, the high number of missed targets, and the increased numbers of steps in which the altitude is increased or decreased.

The macro-averaged metrics were 0.993 for precision, 0.833 for recall and 0.885 for the F1 score. These values indicate that

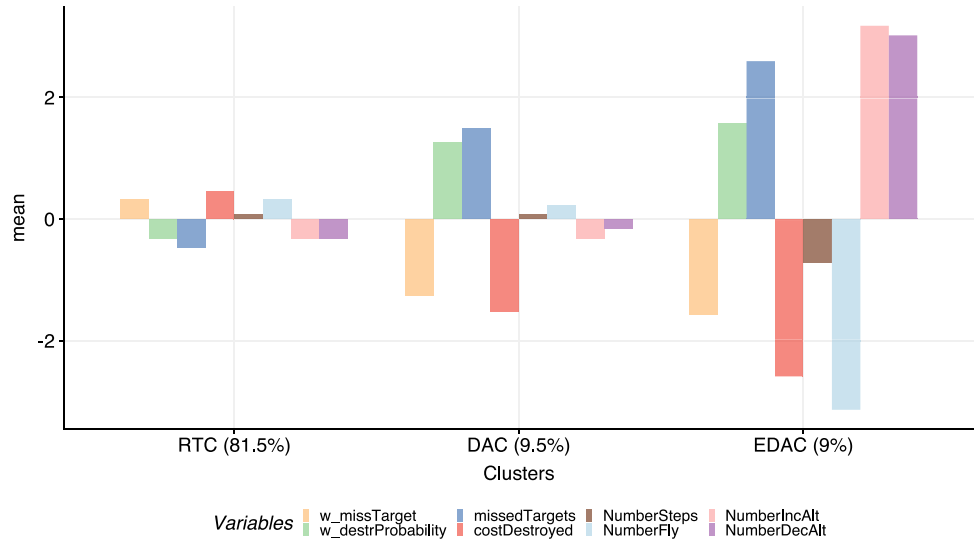


Fig. 13. Bar plot of the cluster means for DARTSim.

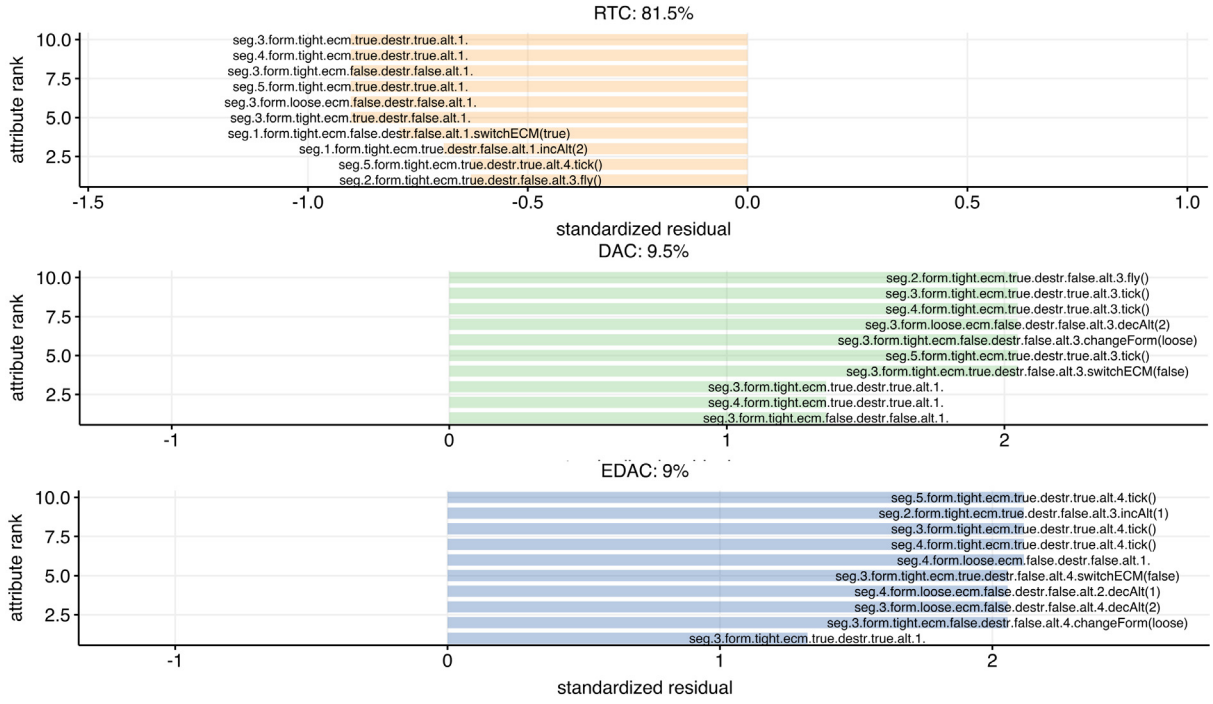


Fig. 14. Variables with the highest standardized residuals (positive or negative) for each cluster in DARTSim.

the clustering based on joint dimensionality reduction and k-means correspond to the clustering based on manual labeling. The average silhouette width value was 0.875. It indicates that the clusters were rather well-separated and that objects in the same cluster are closer to each other than to objects in other clusters (Lengyel and Botta-Dukát, 2019).

These differences in quantitative variable values are also connected to differences in policies and actions being taken. Fig. 14 shows the top ten categorical variables with the highest standardized residuals for the three clusters. Policies in RTC (the risk-taker cluster) have negative standardized residuals for multiple variables with empty values (which indicate that the fleet does not visit states in which the fleet is at an altitude of 1 with ECM switched on). Note that the standardized residuals are negative, which implies that it is common for policies in this cluster to fly at a low altitude and switch ECM off. This observation is consistent with RTC being the risk-taker cluster, because it is riskier to fly at a low altitude with ECM switched off, but it also leads to a higher number of detected targets. Policies in DAC (the destruction-avoidant cluster) tend to fly at an altitude of 3, switch on ECM, and change their form to a loose formation and lower their altitude to 2 in segment 3. This strategy increases the probability of avoiding destruction. Policies in EDAC (the extremely destruction-avoidant) tend to increase and decrease the fleet's altitude a lot and frequently fly at an altitude of 4. This strategy leads to a higher probability of missing a target, but also lowers the probability of being destroyed. One observation when looking at these different strategies is that the decision taken in segment 2 shapes the way for the subsequent steps of the mission: policies in RTC tend to avoid flying in a tight formation with ECM switched on at an altitude of 3, whereas policies in DAC

Table 8
Information reduction and explained variance summary (numerical dataframe, with PCA).

System	# data-frame vars.	# relevant PCA vars.	Information reduction	Explained variance	Residual variance
Robot mission planning	22	20	9.09%	74.67%	25.33%
DART	16	15	6.25%	65.3%	34.7%

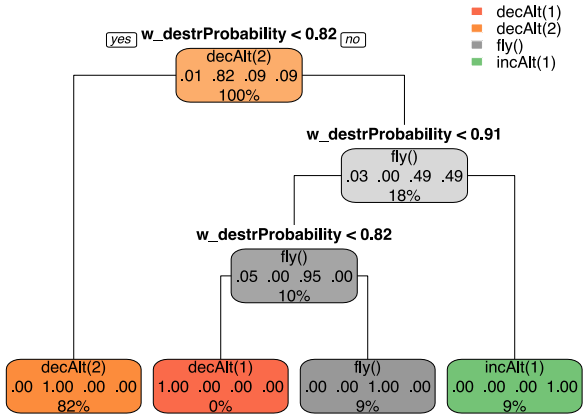


Fig. 15. Decision tree plot for DARTSim (showing the selected decision in the state `seg.2.form.tight.ecm.true.destr.false.alt.3`).

tend to do exactly that, and policies in EDAC tend to be in that state and increase the altitude further.

The previous observation can be further explored using a decision tree depicted in Fig. 15. It can be seen what actions are taken in the segment 2 depending on the selected utility function weights. In this case, the decisions depend on `w_destr Probability`. In the experiment we conducted, all policies visit the state where the fleet is in segment 2 in a tight formation with ECM switched on and at an altitude of 3. For policies with a low weight of avoiding destruction (`w_destrProbability` $<$ 0.82), the decision in this step is to decrease their altitude by 2 (and thus increase the probability of not missing a target). This action is common for policies in RTC (the risk-taker cluster). For policies with a high weight of avoiding destruction (`w_destrProbability` $\geq$ 0.91), the decision is to increase the altitude by 1, which makes it even more unlikely to be hit by a threat. Policies in EDAC tend to adopt this behavior. Policies with a weight of avoiding destruction in the interval $[0.82, 0.91)$ fly to the next segment and keep their altitude. From our previous observations, we can see that this decision is common for policies in DAC.

What can be concluded from the findings described in this section is that the tradeoff explanation approach is applicable to DARTSim. The main tradeoff lies in accepting the risk of potential destruction vs. accepting potentially missed targets. Based on the clustering analysis, we identified that the policies could be categorized as either being part of the risk-taker cluster, the destruction-avoidant cluster, or the extremely destruction-avoidant cluster (that tends to increase its altitude a lot to ensure that threats are avoided). Human stakeholders can leverage our analysis methods to arrive at these conclusions and more significantly identify the thresholds in utility function weights leading to differences in the clusters. In this case, the relevant utility function weight is the weight of avoiding destruction, with relevant thresholds between 0.82 and 0.91. In the following subsection, we describe how the tradeoff explanation approach can be applied to other self-adaptive systems.

6.2.2. Tradeoff between information reduction and explained variance (RQ2)

Tables 8 and 9 summarize the amount of reduced information and explained variance using PCA and MCA for the two systems we used for evaluation. The tables indicate the number of variables in the dataframe, as well as the relevant variables indicated by PCA and MCA. The relevant variables are those outside the area that indicates 50% of the explained variance in the data (indicated by a circle in the PCA plots). The column "information reduction" indicates the extent of information reduction if only the relevant variables are considered. Explained variance is the sum of the percentages that the first two principal components/dimensions account for. Residual variance is the remaining variance that is not explained by the first two components/dimensions.

From the tables, it can be seen that the reduction in the information that needs to be processed by human stakeholders is greatest for the categorical dataframe used for MCA, i.e., 28.57% and 56.25%. In the case of PCA, many dataframe variables were filtered out before applying PCA because they had constant or empty values, leading to an information reduction of less than 10%. The tables also indicate that the first dimensions/components explain a large amount of the variance in the data, with a residual variance ranging between 22.19% and 43.91%. To decrease the residual variance further, it can be useful to consider a third principal component or dimension in the analysis. While 3D PCA plots can be generated, considering more dimensions makes it more difficult to create illustrative plots for analysis. Overall, the observations regarding information reduction and explained variance indicate that the approach supports considerable information reduction, while maintaining a large amount of explained variance. The combination of dimensionality reduction techniques with clustering and decision tree learning can further facilitate the focused analysis of tradeoffs, planning strategies, and thresholds and reduce the information that needs to be processed by human stakeholders.

Note that the information reduction obtained by PCA/MCA is not the only benefit of our approach. The use of clustering techniques also reduces the amount of information an expert needs to analyze: instead of having to deal with all samples, it is sufficient to analyze the characteristics of clusters/policy strategies. DTL is especially useful to get insights into variable thresholds impacting the generation of plans with many states and actions. The example applications of robot planning and DARTSim have large state–action spaces. For web-based systems with few states, however, we expect the information reduction achieved by decision tree learning to be less substantial.

6.3. Threats to validity

In the following, we discuss threats to validity:

Internal validity. This type of validity is concerned with confounding factors influencing quality tradeoff explanations that we did not consider in this research. The feasibility of applying our approach could have been influenced by the concrete sampling strategy (i.e., uniform sampling) that we applied when collecting data or by specific characteristics of the systems under study. To be transparent about potentially confounding factors, we described the systems and method in detail.

Table 9
Information reduction and explained variance summary (categorical dataframe, with MCA).

System	# data-frame vars.	# relevant MCA vars.	Information reduction	Explained variance	Residual variance
Robot mission planning	35	25	28.57%	77.81%	22.19%
DART	80	35	56.25%	56.09%	43.91%

Conclusion validity. Although we do not present any statistically significant conclusions in this paper, we might have missed important findings or presented findings that do not hold in practice. To account for threats to conclusion validity, we present a clear chain of evidence when presenting our method and findings. For the feasibility of our approach, we describe our findings in detail and trace them to the generated plots. For the enabled information reduction, we quantify our findings. We also provide example dataframes and plots in our Github repository.

Construct validity. The constructs that are relevant in our study are subject to different interpretations. For example, the construct of feasibility (RQ1) could be understood in different ways. We refer to the extent to which it is possible to generate explanations for automated planning systems and draw tradeoff-related conclusions from them. We describe the conclusions we draw in detail and aim to provide a chain of evidence from the conclusions to the data in the plots.

Reliability. Reliability is connected to the researchers impacting conclusions presented in this paper. Given that there is no standard way of interpreting and describing plots, certain insights might be dependent on our interpretation and description. We provide a replication package to enable others to revisit our conclusions and apply our approach to other systems.

External validity. One threat to external validity is that our presented findings might not generalize to other systems and contexts. We selected two systems to evaluate our findings, with one being concerned with robot mission planning and the other one with planning for a fleet of drones. For these systems, our approach was feasible and led to a considerable information reduction while preserving relevant information. Other contexts and systems might come with varying degrees of feasibility and information reduction. In Section 8, we discuss the expected levels of feasibility for a set of self-adaptive exemplars.

7. Related work

Explainability for self-adaptive systems. The need for explainable approaches for self-adaptive systems has been previously identified (Bencomo et al., 2012). In particular, our proposed approach can help practitioners assess why an adaptation strategy is chosen and what the adaptation space can be characterized by Diallo et al. (2021) presented an explainable framework relying on convolutional neural networks to reduce the adaptation space. The focus of their approach is not on discerning and explaining quality attribute tradeoffs, but rather on adaptation space reduction, which is related to the employed dimensionality reduction techniques in our approach.

Given that humans should not be overwhelmed with information, an approach has been designed to identify when to provide explanations to users (Li et al., 2020). The authors found that explanation can improve a system's performance, especially for human users with intermediate training levels. The presented approach is agnostic with respect to the mechanisms employed for explanation and thus can complement our approach, providing insights about how it can be exploited in the best possible way in some contexts.

Run-time goal-based models have been used as a basis to create natural language explanations about systems' run-time behaviors (Welsh et al., 2014). Another approach relies on provenance graphs to collect data from a system at run time and explain it to users (Reynolds et al., 2020). Similarly, historical data can be used to provide explanations to users, either after the system has finished running or as live explanations (Garcia Dominguez et al., 2019). These approaches can be used to create an explanation for a particular execution of a system at a point in time. Our approach provides a high-level explanation of quality tradeoffs in the design space based on data. To the best of our knowledge, this high-level explanation based on machine learning techniques has not been covered by previous approaches.

Explainability of tradeoff spaces. In the field of software architecture, an approach to explaining architectural design tradeoff spaces has been developed that relies on PCA to support human designers in analyzing tradeoffs (e.g., between cost, reliability, and performance) (Cámara et al., 2021). Our method is based on similar ideas, but extends the scope and incorporates other ML-based methods (i.e., MCA, clustering, and DTL) suitable for tradeoff analysis.

Another related approach is focused on tradeoff-focused contrastive explanation for MDP planning, which involves contrasting a selected policy to Pareto-optimal alternative planning solutions and arguing about their impact on quality attributes (Sukkerd et al., 2020). A similar approach relies on contrastive explanations for MDP planning, which focuses on describing "critical states" and the impact of decisions on the flexibility to replan the route at run time (Chen et al., 2020). In contrast to these approaches, our research focuses on quality attributes and strategies (i.e., clusters of policies sharing similar characteristics) to explain the tradeoff space at a high level.

Explainable AI. In artificial intelligence, there is a growing body of work in the emerging area of *eXplainable AI* (also referred to as XAI and *interpretable AI* in the literature) that aims at creating techniques that can yield more understandable models that enable humans to understand, appropriately trust, and effectively manage emerging AI-based systems. However, the field is constrained to black-box machine learning systems and a recent extensive survey of the area does not show any XAI approaches that target automated planning (Barredo Arrieta et al., 2020). One approach in the area of XAI supports the extraction of history-aware explanations on demand when using reinforcement learning, including data on measurements and quality attributes (Parra-Ullauri et al., 2021). While it does not focus on automated planning, it is similar to our approach since historical data is leveraged to generate explanations.

Explaining decisions in reinforcement learning. To explain the decisions taken in reinforcement learning, an approach has been developed that involves learning decomposed reward function components for an MDP and using them to predict the expected rewards in different quality attribute dimensions (Juozapaitis et al., 2019). This approach learns the optimal policy at the same time as the explanations (and the reward dimensions that the system tries to maximize). Our approach is similar in the sense that it helps to discern and explain different strategies with different quality attribute characteristics.

The explanation of actions through hierarchical goals is another direction in explainable AI. The Dot-to-Dot method (Beyret

Table 10
Exemplars and inputs required to apply our approach.

Domain (exemplars)	Variables (environment)	Variables (policies)	Quality attributes
Web/Cloud/Service-based Znn.com (Cheng et al., 2009), Hogna (Barna et al., 2015), TAS (Weyns and Calinescu, 2015), Hadoop-Benchmark (Zhang et al., 2017), CrowdNav and RTX (Schmid et al., 2017), mRUBiS (Vogel, 2018), K8-Scalar (Delnat et al., 2018), SWIM (Moreno et al., 2018), OCCI Monitoring (Erbel et al., 2019), RDMSim (Samin et al., 2021)	Request arrival rate, service reliability, service availability, network latency	Server/virtual machine pool size, content fidelity, maximum service invocation retries, timeout length, service selection, allocation of resources	Performance (response time, throughput), cost, resource consumption, content fidelity, availability, reliability (e.g., number of failed service invocations, mean time to recovery)
Autonomous Vehicles/Robotics ATRP (Wuttke et al., 2012), Dragonfly (Maia et al., 2019), DARTSim (Moreno et al., 2019), UNDERSEA (Gerasimou et al., 2017), TRAPP (Gerostathopoulos and Pournaras, 2019), RoboMAX (Askarpour et al., 2021)	Weather conditions, obstacles, speed limits, traffic accidents, road closures, desired fairness level for planning, sensor reliability, map	Route selection, speed, sensor configuration, planning frequency	timeliness, energy consumption, safety, reliability, robustness, scalability, usability, performance, utilization of resources
Cyber–physical Systems/IoT DEECo (Al Ali et al., 2014), FmFM (Bennaceur et al., 2016), DeltaIoT (Iftikhar et al., 2017), Intelligent Ensembles (Krijt et al., 2017), DingNet (Provoost and Weyns, 2019), AMELIA (Tsigkanos et al., 2019), Platooning LEGOs (Shin et al., 2021), Body Sensor Network (Gil et al., 2021)	Traffic load, communication interference, sensor reliability, map, resources	Network settings (e.g., transmission power, spreading factor), assignment of resources, route selection	reliability, cost, energy efficiency, travel speed, utilization of resources, security

et al., 2019) uses hierarchical reinforcement learning with hindsight experience replay for robotic manipulation and breaks the mission goal into smaller sub-goals whose rewards shall be maximized. The high-level representation can be presented to humans and interpreted more easily, which is in line with our approach that helps to abstract from the state–action pairs in policies and describes them as clusters of strategies.

While the explanations provided by our method are global explanations at a high level of abstraction (Chakraborti et al., 2020), they also allow users to investigate local decisions using DTL. We are not aware of any quality tradeoff explanation approaches for self-adaptive systems with these properties.

Defining utility function weights. In the context of automated planning, our previous work presents a tool-supported negotiation technique for preference and constraint elicitation, whose input is used to define the weights of a utility function (Wohlrab and Garlan, 2021a,b). While the approach helps stakeholders come to an agreement about utility function weights, users are unable to determine what the consequences on system behavior are. This paper addresses this issue by providing an explainability approach that can help humans identify how to best select utility function weights and achieve the desired adaptation behavior.

8. Discussion

Feasibility (RQ1). The results of applying our approach to the robot mission planning and DARTSim systems indicate that our approach is applicable to extract information regarding quality tradeoffs, strategies, and defining variable thresholds (e.g., in utility function weights). Our findings are consistent with our observations when examining models and simulation results of the systems.

To further evaluate the feasibility of our approach, we systematically went through the set of exemplars for self-adaptive systems curated by the Software Engineering for Adaptive and Self-Managing Systems (SEAMS) community. The community supports a curated repository of example systems and problems that can be used as a motivation for research, to showcase and assess solutions and techniques, and to compare results.[5] We assessed what pieces of data would have to be extracted and what additional steps are required to apply our approach to these systems.

Table 10 gives an overview of our findings. Many of these exemplars exhibit a high degree of commonality, so we consider the categorization on the exemplar website, namely: web/cloud/service-based systems, autonomous vehicles/robotics systems, and cyber–physical/IoT systems. For each category, we indicate the key environmental variables to consider, the variables that our approach can collect from each generated policy, and the quality attributes that are affected by the policies.

Adaptations in the domain of web/cloud/service-based systems are affected by environmental variables such as the number of incoming requests per time unit, network latency, service failure rates, and services becoming (un)available (e.g., in TAS). Their adaptation policies are generally concerned with changing the server (e.g., Znn.com, mRUBiS, SWIM) or virtual machine (e.g., Hogna, K8-Scalar) pool size and the fidelity of the contents served (e.g., text vs. multimedia mode in Znn.com, percentage of requests served with additional content in SWIM). Relevant quality attributes include performance, resource consumption, cost, content fidelity, availability, and reliability.

Autonomous vehicles and (mobile) robotic systems are affected by environmental variables that are often related to the physical environment in which the system is operating (e.g., weather conditions, obstacles, e.g., in Dragonfly) and certain given characteristics of the planning problem (e.g., whether the plans should be optimized for fairness, e.g., in TRAPP). Policies generally indicate the routes to a target destination (e.g., in ATRP

[5] https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/

or RoboMAX), the sensor configuration and speed (e.g., in UNDERSEA), and the frequency at which planning shall be performed (e.g., in TRAPP). The quality attributes to consider are related to the timeliness, energy consumption, safety aspects (e.g., collision avoidance), reliability, robustness, scalability, usability (e.g., in RoboMAX), performance, and the utilization of resources (e.g., streets in TRAPP).

Cyber–physical/IoT systems need to consider a variety of environmental factors, depending on the concrete application context (e.g., traffic load, communication interference, or sensor reliability, map, or the existence of resources). Policies are concerned with the assignment of parking spaces or other resources (e.g., in DEECo or Intelligent Ensembles), adjusting the network settings (e.g., in DingNet), routes, or the number and timing of lane changes and speed adjustments in Platooning LEGOs. Quality attributes range from reliability (e.g., the number of successfully delivered packages in DeltaIoT), cost, and energy efficiency, to the travel speed, utilization of resources, and security concerns (e.g., in DingNet).

Several exemplars rely on utility or cost functions for self-adaptation, i.e., UNDERSEA, TRAPP, and certain implementations of DARTSim, RoboMAX, and DeltaIoT, as well as Znn.com, mRUBis, and SWIM. Other exemplars use goal models (e.g., FmFm) or implicit representations of the quality attributes to optimize for and the constraints to meet. Although our tradeoff explanation approach primarily focuses on systems that are based on utility functions, it is also potentially of use to applications that consider multiple (competing) quality attributes. We consider the approach less useful if only one quality attribute is optimized by a system. However, the overview of the exemplars indicates that there exists an inherent set of quality attributes for all exemplar categories which should be considered by realistic systems. Systems relying on only one quality attribute appear to be less common and useful in practice.

When collecting data and running our tradeoff explanation approach, certain techniques employed in the approach might be more suitable to particular exemplars. For exemplars whose policies can be characterized by quantitative values (e.g., for network settings, planning frequency, pool size, timeout length, and invocation retries), the most informative findings can be reached when using PCA (instead of MCA) for joint dimensionality reduction and clustering. In those cases, when creating a decision tree, stakeholders might be mainly interested in understanding *threshold* values characterizing different policies and what they depend on.

For exemplars whose policies are best described using categorical variables (e.g., routes, allocation of resources, and service selection), it is advisable to use MCA for dimensionality reduction. Aspects to focus on are *decisions* that differentiate one cluster of policies from another (e.g., in route planning).

For the exemplar categories of autonomous vehicles/robotics and cyber–physical systems/IoT, we expect a larger and more heterogeneous set of variables to be collected than for web/cloud/service-based systems. Environmental variables are less easy to capture for systems that need to take the physical world into consideration than for web/cloud/service-based applications. Taking weather conditions as an example, one can imagine a variety of sensors capturing various aspects that might be relevant (e.g., temperature, cloudiness, pressure, precipitation, and wind). An important step when applying our approach is to filter out variables that do not account for a large amount of the explained variance and reduce the dimensionality of the adaptation space.

With these insights in mind, researchers and practitioners in the future can collect data in a targeted way and apply our tradeoff explanation approach to other systems. Since we do not expect the approach to be applicable as a one-size-fits-all solution, we described the aspects requiring human input in this section, including the focus on PCA/MCA, variable selection for decision tree learning, and the variables to focus on during data collection. The fact that humans might need to give input to create dataframes and select variables of interest constitutes a potential limitation of our approach. It is reasonable to expect stakeholders to be able to provide the necessary inputs, given that the general problem domain and potentially interesting quality attributes should be known. Since the first steps of the approach (PCA and MCA) provide a high-level overview of variables and correlations, these insights can be leveraged when diving into the details and creating decision trees. However, from our experiences with several planning domains, we realized that it can be difficult to correctly interpret the results if the semantics of a variable is unclear. In these situations, it can help to explore the policy generation and planning problem in further detail to better understand the generated plots and explanations.

Our assessment of the applicability of our method to the SEAMS exemplars identified the variables that need to be extracted, aspects to consider when tailoring the approach and giving human input, and the use of utility functions. Our findings show that not all exemplars rely on explicit utility functions, and whether PCA or MCA is most adequate for dimensionality reduction depends on the nature of the exemplar. While not all exemplars employ utility functions, the systems we focused on in this paper (robot mission planning and DARTSim) and many others that require multiple quality attributes to be traded off against each other do employ them. The weighted sum approach is one that is commonly applied in related works (Ghezzi and Molzam Sharifloo, 2013; Cheng et al., 2006; Esfahani et al., 2013; Sousa et al., 2008). Even for systems not relying on utility functions, our approach is applicable, as long as there are two or more quality attributes of interest that can be traded off against each other. Besides the weighted sum model, the weighted product model is a common approach that also relies on utility function weights (Triantaphyllou, 2000). Generally speaking, for any utility functions that encode weights or preference orders, our approach is likely to produce informative results. The primary assumption that we make is that you can generate a variety of points in the design space to explore the planning space. Future work can investigate the applicability of our approach to other kinds of utility functions.

Information reduction (RQ2). For our example systems, we identified an information reduction of 29%–56% with an explained variance in the range of 56%–78%. These findings indicate that there is a substantial level of information reduction while a large amount of the explained variance is retained. Note that the dimensionality reduction techniques PCA and MCA are not the only component of our method, but are complemented with clustering algorithms and decision tree learning. Clustering algorithms help to further improve the insights users can get by describing policies in terms of two to four categories for these example systems. Decision tree learning supports understanding, e.g., by indicating how values of utility function weights impact the actions to be chosen.

Discussion of the approach's evaluation and applicability in practice. On a conventional laptop, the data generation took 42.93s for the robot mission planning example with the dataframe dimensions indicated in Table 5. Once the data has been generated, executing the script to create plots took 20.23s for the robot mission planning example with a conventional laptop. For even larger problems or higher values for the optimal number of clusters, the execution time would be even longer. While the approach is not very time-consuming, we do not envision this approach to be applied continuously at run time. Rather, we intend it to be

used to train stakeholders in understanding tradeoffs and helping them to indicate their preferences appropriately to deliberately set utility function weights. Currently, it is often obscure what the impact on the planned behavior is when defining utility functions. Our approach aims to address this issue.

The current evaluation focuses on systems with up to three quality attributes. When scaling the approach to larger systems with more quality attributes, the sampling strategy to collect data would likely need to be revised. We applied uniform sampling of different combinations of utility function weights. Depending on the planner, the complexity of the problem, and the number of quality attributes, the sampling should be performed in a more coarse-granular way. We recommend starting with coarse-grained sampling and performing complementary sampling around key threshold regions. Future work will identify appropriate data collection strategies for practical contexts.

The approach's advantages are its applicability to large, complex design spaces and the large reduction of information (RQ2). The approach's disadvantage in its current version is the required level of expertise needed to interpret and understand the plots. While we found the conclusions to be understandable with adequate additional explanations, further research is needed to create visualizations or natural language explanations that can be used by stakeholders of various backgrounds and disciplines. In the future, these approaches can be evaluated with respect to how fast and accurately humans can assess explanations. For example, glitch detector tasks (Hoffman et al., 2018) can be used to investigate humans' mental models by asking them to find glitches/mistakes in explanations.

9. Conclusions

In this paper, we have presented an approach to explain quality attribute tradeoffs in automated planning for self-adaptive systems. Our approach relies on machine learning methods to describe the relevance of quality attributes for automated planning, their relations, strategies (i.e., groups of planning policies sharing similar characteristics), as well as key thresholds in utility function weights and their impact on generated policies. We evaluated the approach with respect to feasibility (RQ1) by applying it to two systems (robot mission planning and DARTSim) and describing the potential application to 24 exemplar self-adaptive systems. We also described the considerable level of information reduction when applying our approach and the moderate reduction in explained variance (RQ2).

We observed a number of limitations of our approach, for instance, with respect to the use of PCA for joint dimensionality reduction and clustering. PCA is most applicable when correlations between variables are linear and would need to be replaced with other techniques when non-linear correlations are of interest.

An area of future work is to explore alternative sampling strategies. The sampling approach used to generate adaptation policy data might influence the conclusions that are reached. For the examples described in this paper, we sampled uniformly over the space of parameters (i.e., the utility weights w_i in the ranges $[0, 1]$). To be able to better approximate thresholds, e.g., for decision-tree learning, finer-grained sampling around key threshold regions could be beneficial. Future work will focus on alternative sampling strategies based on previous work such as cross-entropy methods (Moreno et al., 2017).

Finally, while the insights that can be obtained from applying our approach can help human stakeholders and system designers better understand the otherwise untamable complexity of automated planning, the generated plots might be difficult to grasp for untrained users. To address this issue, we plan to develop tools with appropriate user interfaces that can visualize and explain tradeoffs in automated planning and help stakeholders provide input to ensure that self-adaptive systems' plans meet their requirements (e.g., by selecting appropriate values for utility function weights).

CRediT authorship contribution statement

Rebekka Wohlrab: Methodology, Conceptualization, Investigation, Formal analysis, Resources, Visualization, Project administration, Software, Writing – original draft, Writing – review & editing. **Javier Cámara:** Methodology, Conceptualization, Investigation, Formal analysis, Resources, Visualization, Writing – original draft, Writing – review & editing. **David Garlan:** Methodology, Conceptualization, Resources, Writing – review & editing. **Bradley Schmerl:** Methodology, Conceptualization, Resources, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

We have uploaded the data/code to a repository that is linked in the paper.

Acknowledgments

This work is supported in part by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, by award N00014172899 from the Office of Naval Research and by the NSA under Award No. H9823018D000. This work was also partially supported by the Spanish Government (FEDER/Ministerio de Ciencia e Innovación – Agencia Estatal de Investigación) under project COSCA (PGC2018-094905-B-I00). Any views, opinions, findings and conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding sources.

References

Al Ali, R., Bures, T., Gerostathopoulos, I., Hnetynka, P., Keznikl, J., Kit, M., Plasil, F., 2014. DEECo: An ecosystem for cyber–physical systems. In: Proceedings of the 36th International Conference on Software Engineering. ICSE-C pp. 610–611. http://dx.doi.org/10.1145/2591062.2591140.

Askarpour, M., Tsigkanos, C., Menghi, C., Calinescu, R., Pelliccione, P., García, S., Caldas, R., von Oertzen, T.J., Wimmer, M., Berardinelli, L., Rossi, M., Bersani, M.M., Rodrigues, G.S., 2021. RoboMAX: Robotic mission adaptation exemplars. In: Proceedings of the 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 245–251. http://dx.doi.org/10.1109/SEAMS51251.2021.00040.

Barna, C., Ghanbari, H., Litoiu, M., Shtern, M., 2015. Hogna: A platform for self-adaptive applications in cloud environments. In: Inverardi, P., Schmerl, B.R. (Eds.), Proceedings of the 10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS pp. 83–87. http://dx.doi.org/10.1109/SEAMS.2015.26.

Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., Herrera, F., 2020. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible Ai. Inf. Fusion 58, 82–115. http://dx.doi.org/10.1016/j.inffus.2019.12.012.

Batool, F., Hennig, C., 2021. Clustering with the average silhouette width. Comput. Statist. Data Anal. 158, 107190.

Bencomo, N., Welsh, K., Sawyer, P., Whittle, J., 2012. Self-explanation in adaptive systems. In: Proceedings of the IEEE 17th International Conference on Engineering of Complex Computer Systems. pp. 157–166. http://dx.doi.org/10.1109/ICECCS20050.2012.6299211.

Bennaceur, A., McCormick, C., Galán, J.G., Perera, C., Smith, A., Zisman, A., Nuseibeh, B., 2016. Feed me, feed me: An exemplar for engineering adaptive software. In: Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 89–95. http://dx.doi.org/10.1145/2897053.2897071.

Beyret, B., Shafti, A., Faisal, A.A., 2019. Dot-to-dot: Explainable hierarchical reinforcement learning for robotic manipulation. In: Proceedings of the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS, pp. 5014–5019. http://dx.doi.org/10.1109/IROS40897.2019.8968488.

Bianco, A., De Alfaro, L., 1995. Model checking of probabilistic and nondeterministic systems. In: Proceedings of the International Conference on Foundations of Software Technology and Theoretical Computer Science. Springer pp. 499–513.

Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J., 2017. Classification and Regression Trees. Routledge.

Cámara, J., Schmerl, B., Garlan, D., 2020. Software architecture and task plan co-adaptation for mobile service robots. In: Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 125–136. http://dx.doi.org/10.1145/3387939.3391591.

Cámara, J., Silva, M., Garlan, D., Schmerl, B., 2021. Explaining architectural design tradeoff spaces: A machine learning approach. In: Proceedings of the 15th European Conference on Software Architecture. ECSA, Springer International Publishing, Cham, pp. 49–65.

Chakraborti, T., Sreedharan, S., Kambhampati, S., 2020. The emerging landscape of explainable automated planning & decision making. In: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence. IJCAI-20, pp. 4803–4811.

Chen, S., Boggess, K., Feng, L., 2020. Towards transparent robotic planning via contrastive explanations. In: Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS, IEEE, pp. 6593–6598.

Cheng, S.-W., Garlan, D., Schmerl, B., 2006. Architecture-based self-adaptation in the presence of multiple objectives. In: Proceedings of the 2006 International Workshop on Self-Adaptation and Self-Managing Systems. SEAMS, pp. 2–8.

Cheng, S., Garlan, D., Schmerl, B.R., 2009. Evaluating the effectiveness of the rainbow self-adaptive system. In: Proceedings of the 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 132–141. http://dx.doi.org/10.1109/SEAMS.2009.5069082.

Dehnert, C., Junges, S., Katoen, J.-P., Volk, M., 2017. A storm is coming: A modern probabilistic model checker. In: Proceedings of the International Conference on Computer Aided Verification. Springer, pp. 592–600.

Delnat, W., Truyen, E., Rafique, A., Landuyt, D.V., Joosen, W., 2018. K8-scalar: A workbench to compare autoscalers for container-orchestrated database clusters. In: Andersson, J., Weyns, D. (Eds.), Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 33–39. http://dx.doi.org/10.1145/3194133.3194162.

Diallo, A.B., Nakagawa, H., Tsuchiya, T., 2021. Adaptation space reduction using an explainable framework. In: Proceedings of the IEEE 45th Annual Computers, Software, and Applications Conference. COMPSAC, pp. 1653–1660. http://dx.doi.org/10.1109/COMPSAC51774.2021.00247.

Erbel, J., Brand, T., Giese, H., Grabowski, J., 2019. Occi-compliant, fully causal-connected architecture runtime models supporting sensor management. In: Proceedings of the 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS pp. 188–194. http://dx.doi.org/10.1109/SEAMS.2019.00032.

Esfahani, N., Elkhodary, A., Malek, S., 2013. A learning-based framework for engineering feature-oriented self-adaptive software systems. IEEE Trans. Softw. Eng. 39 (11), 1467–1493.

García, S., Menghi, C., Pelliccione, P., 2019. Mapmaker: performing multi-robot LTL planning under uncertainty. In: Proceedings of the 2019 IEEE/ACM 2nd International Workshop on Robotics Software Engineering. RoSE, IEEE pp. 1–4.

Garcia Dominguez, A., Bencomo, N., Parra Ullauri, J.M., Garcia Paucar, L.H., 2019. Towards history-aware self-adaptation with explanation capabilities. In: Proceedings of the IEEE 4th International Workshops on Foundations and Applications of Self* Systems. FAS*W, pp. 18–23. http://dx.doi.org/10.1109/FAS-W.2019.00018.

Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B., Steenkiste, P., 2004. Rainbow: Architecture-based self-adaptation with reusable infrastructure. Computer 37 (10), 46–54.

Gerasimou, S., Calinescu, R., Shevtsov, S., Weyns, D., 2017. UNDERSEA: An exemplar for engineering self-adaptive unmanned underwater vehicles. In: Proceedings of the 12th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 83–89. http://dx.doi.org/10.1109/SEAMS.2017.19.

Gerostathopoulos, I., Pournaras, E., 2019. TRAPPed in traffic? A self-adaptive framework for decentralized traffic optimization. In: Proceedings of the 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 32–38. http://dx.doi.org/10.1109/SEAMS.2019.00014.

Ghezzi, C., Molzam Sharifloo, A., 2013. Dealing with Non-Functional Requirements for Adaptive Systems Via Dynamic Software Product-Lines. Springer Berlin Heidelberg, pp. 191–213, Ch. 8.

Gil, E.B., Caldas, R., Rodrigues, A., da Silva, G.L.G., Rodrigues, G.N., Pelliccione, P., 2021. Body sensor network: A self-adaptive system exemplar in the healthcare domain. In: Proceedings of the 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS pp. 224–230. http://dx.doi.org/10.1109/SEAMS51251.2021.00037.

Hansson, H., Jonsson, B., 1994. A logic for reasoning about time and reliability. Form. Asp. Comput. 6 (5), 512–535.

Hoffman, R., Mueller, S., Klein, G., Litman, J., 2018. Metrics for explainable AI: Challenges and prospects. XAI Metrics.

Iftikhar, M.U., Ramachandran, G.S., Bollansée, P., Weyns, D., Hughes, D., 2017. DeltaIoT: A self-adaptive internet of things exemplar. In: Proceedings of the 12th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 76–82. http://dx.doi.org/10.1109/SEAMS.2017.21.

Jamshidi, P., Cámara, J., Schmerl, B., Käestner, C., Garlan, D., 2019. Machine learning meets quantitative planning: Enabling self-adaptation in autonomous robots. In: Proceedings of the 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 39–50. http://dx.doi.org/10.1109/SEAMS.2019.00015.

Jolliffe, I.T., 1986. Principal components in regression analysis. In: Principal Component Analysis. Springer, pp. 129–155.

Juozapaitis, Z., Koul, A., Fern, A., Erwig, M., Doshi-Velez, F., 2019. Explainable reinforcement learning via reward decomposition. In: Proceedings of the IJCAI/ECAI Workshop on Explainable Artificial Intelligence. pp. 1–7.

Kounev, S., Brosig, F., Huber, N., 2014. The Descartes Modeling Language. Tech. Rep., Institut Für Informatik, Universität Würzburg.

Krijt, F., Jirácek, Z., Bures, T., Hnetynka, P., Gerostathopoulos, I., 2017. Intelligent ensembles - a declarative group description language and java framework. In: Proceedings of the 12th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 116–122. http://dx.doi.org/10.1109/SEAMS.2017.17.

Kwiatkowska, M., Norman, G., Parker, D., 2011. PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (Eds.), Proceedings of the 23rd International Conference on Computer Aided Verification. CAV'11, In: LNCS, vol. 6806, Springer, pp. 585–591.

Lacerda, B., Parker, D., Hawes, N., 2014. Optimal and dynamic planning for Markov decision processes with co-safe LTL specifications. In: Proceedings of the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 1511–1516. http://dx.doi.org/10.1109/IROS.2014.6942756.

Le Roux, B., Rouanet, H., 2009. Multiple Correspondence Analysis. SAGE Publications.

Lengyel, A., Botta-Dukát, Z., 2019. Silhouette width using generalized mean—a flexible method for assessing clustering efficiency. Ecol. Evol. 9 (23), 13231–13243. http://dx.doi.org/10.1002/ece3.5774.

Li, N., Camara, J., Garlan, D., Schmerl, B., 2020. Reasoning about when to provide explanation for human-involved self-adaptive systems. In: Proceedings of the IEEE International Conference on Autonomic Computing and Self-Organizing Systems. ACSOS, pp. 195–204. http://dx.doi.org/10.1109/ACSOS49614.2020.00042.

Maia, P.H., Vieira, L., Chagas, M., Yu, Y., Zisman, A., Nuseibeh, B., 2019. Dragonfly: A tool for simulating self-adaptive drone behaviours. In: Proceedings of the 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 107–113. http://dx.doi.org/10.1109/SEAMS.2019.00022.

Markos, A., Iodice D'Enza, A., van de Velden, M., 2019. Beyond tandem analysis: Joint dimension reduction and clustering in r. J. Stat. Softw. 91 (10).

Moreno, G., Kinneer, C., Pandey, A., Garlan, D., 2019. DARTSim: An exemplar for evaluation and comparison of self-adaptation approaches for smart cyber–physical systems. In: Proceedings of the 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 181–187. http://dx.doi.org/10.1109/SEAMS.2019.00031.

Moreno, G.A., Schmerl, B.R., Garlan, D., 2018. SWIM: An exemplar for evaluation and comparison of self-adaptation approaches for web applications. In: Andersson, J., Weyns, D. (Eds.), Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 137–143. http://dx.doi.org/10.1145/3194133.3194163.

Moreno, G.A., Strichman, O., Chaki, S., Vaisman, R., 2017. Decision-making with cross-entropy for self-adaptation. In: Proceedings of the 2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 90–101. http://dx.doi.org/10.1109/SEAMS.2017.7.

Parra-Ullauri, J.M., García-Domínguez, A., Bencomo, N., Zheng, C., Zhen, C., Boubeta-Puig, J., Ortiz, G., Yang, S., 2021. Event-driven temporal models for explanations-etemox: Explaining reinforcement learning. Software Syst. Model. 1–23.

Pfitzner, D., Leibbrandt, R., Powers, D., 2009. Characterization and evaluation of similarity measures for pairs of clusterings. Knowl. Inf. Syst. 19 (3), 361–394.

Provoost, M., Weyns, D., 2019. DingNet: A self-adaptive internet-of-things exemplar. In: Proceedings of the 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 195–201. http://dx.doi.org/10.1109/SEAMS.2019.00033.

R. Core Team, 2020. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, URL https://www.R-project.org/.

Reynolds, O., García-Domínguez, A., Bencomo, N., 2020. Automated provenance graphs for models@run.time. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems. MODELS, pp. 1–10.

Samin, H., Bencomo, N., Sawyer, P., 2022. Decision-making under uncertainty: B aware of your priorities. Software Syst. Model. 1–30.

Samin, H., Paucar, L.H.G., Bencomo, N., Hurtado, C.M.C., Fredericks, E.M., 2021. RDMSim: An exemplar for evaluation and comparison of decision-making techniques for self-adaptation. In: Proceedings of the 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 238–244. http://dx.doi.org/10.1109/SEAMS51251.2021.00039.

Schmid, S., Gerostathopoulos, I., Prehofer, C., Bures, T., 2017. Model problem (CrowdNav) and framework (RTX) for self-adaptation based on big data analytics (artifact). Dagstuhl Artifacts Ser. 3 (1), 5:1–5:3. http://dx.doi.org/10.4230/DARTS.3.1.5.

Shin, Y.-J., Liu, L., Hyun, S., Bae, D.-H., 2021. Platooning LEGOs: An open physical exemplar for engineering self-adaptive cyber–physical systems-of-systems. In: Proceedings of the 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 231–237. http://dx.doi.org/10.1109/SEAMS51251.2021.00038.

Sousa, J.P., Balan, R.K., Poladian, V., Garlan, D., Satyanarayanan, M., 2008. User guidance of resource-adaptive systems. In: Proceedings of the 3rd International Conference on Software and Data Technologies. ICSOFT pp. 36–44.

Starnes, D.S., Yates, D., Moore, D.S., 2010. The Practice of Statistics. Macmillan.

Sukkerd, R., Simmons, R., Garlan, D., 2020. Tradeoff-focused contrastive explanation for mdp planning. In: Proceedings of the 29th IEEE International Conference on Robot and Human Interactive Communication. RO-MAN, IEEE, pp. 1041–1048.

Triantaphyllou, E., 2000. Multi-Criteria Decision Making Methods. Springer US, Boston, MA, pp. 5–21, Ch. 2.

Tsigkanos, C., Nenzi, L., Loreti, M., Garriga, M., Dustdar, S., Ghezzi, C., 2019. Inferring analyzable models from trajectories of spatially-distributed internet of things. In: Proceedings of the 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 100–106. http://dx.doi.org/10.1109/SEAMS.2019.00021.

van de Velden, M., Iodice D'Enza, A., Markos, A., 2019. Distance-based clustering of mixed data. Wiley Interdiscip. Rev. Comput. Stat. 11 (3), e1456.

Vogel, T., 2018. mRUBiS: An exemplar for model-based architectural self-healing and self-optimization. In: Andersson, J., Weyns, D. (Eds.), Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 101–107. http://dx.doi.org/10.1145/3194133.3194161.

Welsh, K., Bencomo, N., Sawyer, P., Whittle, J., 2014. Self-explanation in adaptive systems based on runtime goal-based models. In: Transactions on Computational Collective Intelligence XVI. Springer, pp. 122–145.

Weyns, D., Calinescu, R., 2015. Tele assistance: A self-adaptive service-based system exemplar. In: Inverardi, P., Schmerl, B.R. (Eds.), Proceedings of the 10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 88–92. http://dx.doi.org/10.1109/SEAMS.2015.27.

Wohlrab, R., Garlan, D., 2021a. Defining utility functions for multi-stakeholder self-adaptive systems. In: Proceedings of the 27th International Working Conference on Requirement Engineering: Foundation for Software Quality. REFSQ, Springer International, pp. 116–122.

Wohlrab, R., Garlan, D., 2021b. A negotiation support system for defining utility functions for multi-stakeholder self-adaptive systems. Requir. Eng..

Wuttke, J., Brun, Y., Gorla, A., Ramaswamy, J., 2012. Traffic routing for evaluating self-adaptation. In: Müller, H.A., Baresi, L. (Eds.), Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 27–32. http://dx.doi.org/10.1109/SEAMS.2012.6224388.

Zhang, B., Krikava, F., Rouvoy, R., Seinturier, L., 2017. Hadoop-benchmark: Rapid prototyping and evaluation of self-adaptive behaviors in hadoop clusters. In: Proceedings of the 12th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 175–181. http://dx.doi.org/10.1109/SEAMS.2017.15.



Rebekka Wohlrab is an Assistant Professor at the Department of Computer Science and Engineering at Chalmers University of Technology and an adjunct faculty at Carnegie Mellon University. She received her Ph.D. from Chalmers University of Technology. Her research interests include requirements engineering, software architecture, and self-adaptive systems.



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



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



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