ExTrA: Explaining architectural design tradeoff spaces via dimensionality reduction

Citation for the original published paper (version of record):

Cámara, J., Wohlrab, R., Garlan, D. et al (2023). ExTrA: Explaining architectural design tradeoff spaces via dimensionality reduction. Journal of Systems and Software, 198.
http://dx.doi.org/10.1016/j.jss.2022.111578

(article starts on next page)

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

ExTrA: Explaining architectural design tradeoff spaces via dimensionality reduction[☆]

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

[a] ITIS Software, Universidad de Málaga, Spain
[b] Chalmers | University of Gothenburg, Sweden
[c] School of Computer Science, Carnegie Mellon University, United States of America

ARTICLE INFO

Article history:
Received 29 March 2022
Received in revised form 30 August 2022
Accepted 29 November 2022
Available online 19 December 2022

Keywords:
Tradeoff analysis
Uncertainty
Dimensionality reduction

ABSTRACT

In software design, guaranteeing the correctness of run-time system behavior while achieving an acceptable balance among multiple quality attributes remains a challenging problem. Moreover, providing guarantees about the satisfaction of those requirements when systems are subject to uncertain environments is even more challenging. While recent developments in architectural analysis techniques can assist architects in exploring the satisfaction of quantitative guarantees across the design space, existing approaches are still limited because they do not explicitly link design decisions to satisfaction of quality requirements. Furthermore, the amount of information they yield can be overwhelming to a human designer, making it difficult to see the forest for the trees. In this paper we present ExTrA (**Ex**plaining **Tr**adeoffs of software **A**rchitecture design spaces), an approach to analyzing architectural design spaces that addresses these limitations and provides a basis for explaining design tradeoffs. Our approach employs dimensionality reduction techniques employed in machine learning pipelines like Principal Component Analysis (PCA) and Decision Tree Learning (DTL) to enable architects to understand how design decisions contribute to the satisfaction of extra-functional properties across the design space. Our results show feasibility of the approach in two case studies and evidence that combining complementary techniques like PCA and DTL is a viable approach to facilitate comprehension of tradeoffs in poorly-understood design spaces.

1. Introduction

Designing modern software-intensive systems requires exploring architectural design spaces that are often poorly understood due to the increasing complexity and range of choices that architects have to make (and their potential interactions). To add to the problem, it is often challenging to achieve good designs because of high levels of uncertainty. This uncertainty means it is often difficult to guarantee the correctness of run-time system behavior while striking an acceptable balance among multiple nonfunctional properties when design decisions involve selecting and composing loosely coupled, pre-existing components or services that have different attributes (e.g., performance, reliability, cost). These uncertainties can be induced by e.g., faults, changes in resource availability and network conditions, as well as attacks (Garlan, 2010).

There are multiple approaches that help to automate the search for good architecture designs and that rely on a variety of techniques such as stochastic search and Pareto analysis (Aleti et al., 2009; Bondarev et al., 2007; Martens et al., 2010), as well as quantitative verification (Calinescu et al., 2018; Cámara et al., 2019) that enable architects to explore how the satisfaction of quality of service requirements varies as the value of design parameters and environment variables change. Despite being informative, these approaches do not always make clear why and how architectures were selected because: (i) they do not explicitly link design decisions and environmental factors to the satisfaction of requirements, (ii) they yield vast amounts of data that are not easy to interpret by a human designer, and (iii) results include both useful information and noise that obscures understanding the relationship among variables.

Architects need tools and techniques to help them understand the tradeoffs of complex design spaces and guide them to good designs, enabling them to answer questions such as: Why are these tradeoffs being made, and not others? What are the most important parameters and qualities that are driving the key design decisions? How sensitive is the satisfaction of constraints or the achievement of optimality to a particular set of decisions? Which choices are correlated with others, either positively or negatively?

[☆] Editor: Elena Navarro.
[*] Corresponding author.
E-mail address: jcamara@uma.es (J. Cámara).

https://doi.org/10.1016/j.jss.2022.111578
0164-1212/

Providing such tool support demands investigating questions such as:

(RQ1) How can we link architectural design decisions and quantifiable requirements satisfaction in a way that highlights the most important dependencies among them?

(RQ2) How can we quantify the impact of architectural design decisions on the different qualities across the architectural design space?

(RQ3) How much can we reduce the complexity of the information presented to the architect while preserving most of the relevant design tradeoff information?

This paper explores these questions by introducing an approach to enable the explainability of architectural design spaces that addresses the limitations described above. Our approach employs: (i) a dimensionality reduction technique called principal component analysis (PCA) (Jolliffe, 1986) to help identify the most relevant design variables that contribute to QoS variation across the design space, and (ii) Decision Tree Learning (DTL) (Breiman et al., 2017) to help identify how specific design decisions impact system qualities.

Concretely, our approach consists of: (i) extracting design features and quality metrics of a population of architectural configuration samples generated via synthesis and quantitative verification (Cámara et al., 2019), (ii) applying PCA to determine the main variables that influence the qualities of configurations, as well as to establish a link between design variables (e.g., component selection, topological arrangement, configuration parameter values) and the qualities of the resulting configuration, and (iii) applying DTL to identify the thresholds in design variable values that contribute to meaningful variations in QoS variables.

In Cámara et al. (2021) we introduce a preliminary version of this work on explainability that enables the linking of architectural design decisions and requirements satisfaction employing PCA, and evaluate our results on a Tele Assistance System (TAS) (Weyns and Calinescu, 2015) and a network architecture scenario (Kwiatkowska et al., 2009). In this paper, we extend our approach to also make use of Decision Tree Learning (DTL) to provide insights about the impact for architectural decisions on different system qualities. Moreover, we also expand our demonstration of the approach to two variants of TAS and three variants of the network architecture that enable us to better illustrate how changes in design variants can impact system qualities.

Our results show that the approach is feasible in two case studies and provide evidence that combining complementary techniques like PCA and DTL is a viable approach to facilitate comprehension of tradeoffs in poorly-understood design spaces.

The remainder of this paper is organized as follows: Section 2 presents a motivating example (TAS). Section 3 presents our approach to trade-off space explanation via dimensionality reduction using PCA and DTL. Section 4 demonstrates our approach, whereas Section 5 discusses related work. Section 6 concludes the paper and indicates directions for future work.

2. Motivating scenario: Tele-Assistance System (TAS)

TAS (Weyns and Calinescu, 2015) is a service-based system with the goal of tracking a patient's vital parameters to adapt drug type or dose when needed, and taking actions in case of emergency. TAS combines three service types in a workflow (Fig. 1, left). When TAS receives a request that includes the vital parameters of a patient, its *Medical Service* analyzes the data and replies with instructions to: (i) change the patient's drug type, (ii) change the drug dose, or (iii) trigger an alarm for first responders in case of emergency. When changing the drug type or dose, TAS notifies a local pharmacy using a *Drug Service*, whereas first responders are notified via an *Alarm Service*.

The functionality of each service type in TAS is provided by multiple third parties with different levels of performance (response time), reliability (failure rate), and cost (Fig. 1a). Finding an adequate design for the system entails understanding the tradeoff space by selecting the set of system configurations that satisfy: (i) structural constraints, e.g., the *Drug Service* must not be connected to an *Alarm Service*, (ii) behavioral correctness properties (e.g., the system will eventually provide a response — either by dispatching an ambulance or notifying the pharmacy), and (iii) quality requirements, which can be formulated as a combination of quantitative constraints and optimizations (Fig. 1b).

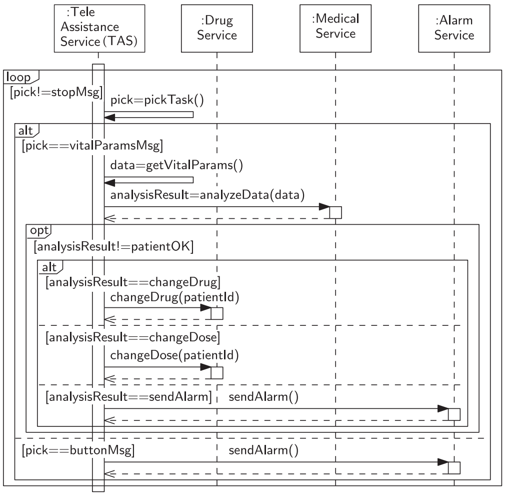
Fig. 2 shows the analysis results of TAS obtained by applying our prior work that combines structural synthesis and quantitative verification to analyze quantitative formal guarantees across the architectural design space (Cámara, 2020; Cámara et al., 2019). The results were generated by automatically synthesizing a probabilistic model of the behavior of each configuration in the system, and then applying probabilistic model checking to quantify the value of the different QoS variables. The plot on the left shows the minimized cost of configurations for different levels of constraints on response time and reliability. This plot conveys the intuition that higher response times and lower reliability correspond to lower costs, whereas peaks in cost are reached with the lowest failure rates and response times.

The plot on the right is a map that shows which configurations best satisfy design criteria. It shows that 24 configurations (out of the 90 possible configurations) satisfy the criteria in some subregion of the state space. If we consider that designers are interested e.g., in systems with response times ≤26 ms and reliability ≥99%, we can employ the same analysis technique to determine which configurations best satisfy these constraints (delimited by the red area in the figure).

Although these plots are informative and can help architects to understand what specific configurations might work well in a given situation, it is not possible to understand what design decisions influence these tradeoffs. Answering to what extent improvements on qualities are a function of the choice of a specific service implementation, the topological arrangement of the composition, or the value of configuration parameters (e.g., maximum number of retries, or timeout duration when services fail) is not possible with existing approaches.

When assessing architectural configurations, it is often hard to tell how specific design decisions result in quality requirements being satisfied to smaller or larger degrees. For instance, there are thresholds in quality measures (e.g., response time) that have to be satisfied. Achieving such quality levels requires making design decisions, choosing among multiple alternatives that entail different costs and trade-offs. This research aims to elicit these thresholds and explain them to architects in order to improve the comprehensibility of the architectural design space.

One of the main challenges in facilitating the understanding of the tradeoff space relates to the high dimensionality of the data and how to convey important factors to a human designer: there are too many characteristics of configurations (and relations among them) to track, and some of them contribute more than others to the variation of quality attributes. For instance, even in the relatively simple system illustrated earlier, it is unclear if selecting specific services contributes more to system quality variation than workflow configuration parameters like timeout length. In the next section, we describe how to address this challenge.



Service name	Fail.rate	Resp.time	Cost
	(%)	(ms.)	(usd)
(S1) Medical S.1	0.06	22	9.8
(S2) Medical S.2	0.1	27	8.9
(S3) Medical S.3	0.15	31	9.3
(S4) Medical S.4	0.25	29	7.3
(S5) Medical S.5	0.05	20	11.9
(AS1) Alarm S.1	0.3	11	4.1
(AS2) Alarm S.2	0.4	9	2.5
(AS3) Alarm S.3	0.08	3	6.8
(D1) Drug S.1	0.12	1	0.1

(a) Properties of TAS service providers.

Id	Description
R1	The average failure rate should not exceed 0.03%.
R2	The average response time should not exceed 26 ms.
R3	Subject to R1 and R2, the cost should be minimized.

(b) Example of quality requirements.

Fig. 1. TAS workflow, service provider properties, and quality requirements.

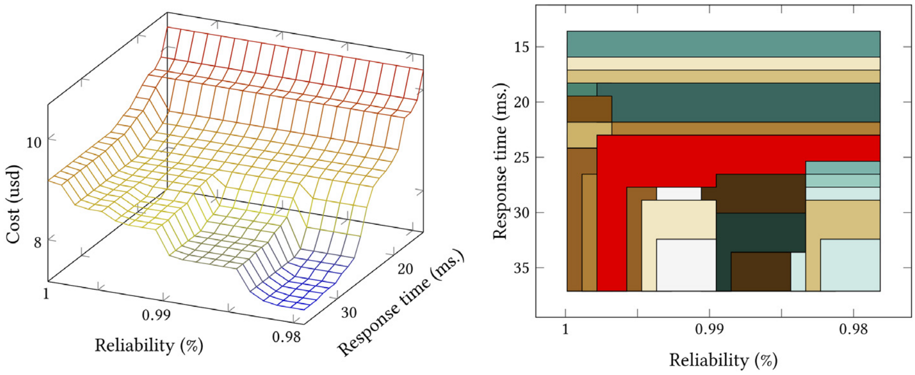


Fig. 2. TAS Analysis results.

3. Approach

The inputs to our approach for explaining design tradeoffs (Fig. 3) are a set of legal configurations (i.e., those that satisfy the constraints of a given architectural style), captured as attribute-annotated graphs, and a set of quantitative metrics that can capture aspects related to e.g., the energy consumption, timeliness, or safety of configurations. The outputs of the process are: (i) a plot that captures a description of the relations between design and QoS variables (e.g., response time is negatively correlated with reliability, selection of component X contributes to lower response times and higher cost), as well as their contributions to differences among architectural configurations, and (ii) a set of decision trees that capture how the selection of values for design variables influences the outcome of the design in terms of the quality concerns considered (e.g., for values of the timeout length parameter below X, the values of response time are constrained to the range [Y,Z]). The approach consists of five stages:

1. *Configuration Data Extraction* collects relevant information about the characteristics of architecture configurations. Data extracted includes both topological information (e.g., centrality and cardinality measures of nodes corresponding to different component types and bindings) and information related to properties of components, connectors, and other parameters.
2. *Data Aggregation and Normalization*. In this step, the configuration data produced in (1) and the configuration metrics provided as input to the process are aggregated into a single dataframe (Stage 2a). Table data is further normalized (Stage 2b) into a correlations table so that all variables will have the same weight in the subsequent principal component analysis (Stage 3).
3. *Principal Component Analysis (PCA)* is employed to discover how architectural configurations differ, and which variables contribute the most to that difference. Moreover, PCA enables us to discriminate whether variables are positively or negatively correlated, or if instead they are independent from each other. This enables architects to relate response variation (QoS, quantitative guarantees) to design variables.
4. *Target Variable Selection* is performed by the architect based on the information obtained from PCA that identifies the most important variables and dependencies in the design space. In this stage, the architect selects the set of relevant variables associated with decisions for which she wants to quantify the impact on different quality concerns.

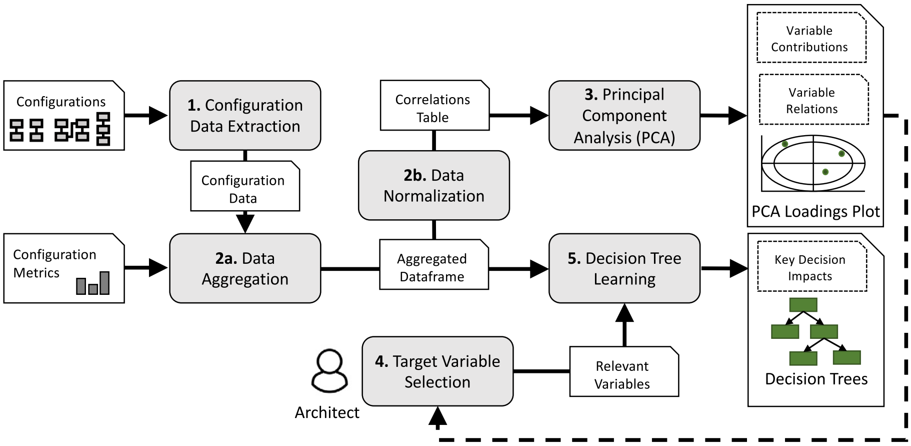


Fig. 3. Overview of the approach.

5. *Decision Tree Learning* (DTL) receives as input both the aggregated dataframe produced in stage 2 and the set of relevant variables identified in stage 4, and produces a set of decision trees that link concrete threshold values of design variables with impacts on qualities that architects can use to inform their decisions. Often, PCA and DTL are used independently and for different purposes. However, in the context of ExTrA, we use the results from PCA analysis to inform the creation of decision trees. This is because the contributions to variability and relations identified by PCA can influence which target variables are selected by the architect (Stage 4) and used as inputs to the creating the decision trees.

In the remainder of this section, we first introduce some preliminaries, and follow with a detailed description of the five stages of our approach.

3.1. Preliminaries

Design spaces are often constrained by the need to design systems within certain patterns or constraints. Architectural styles (Shaw and Garlan, 1996) characterize the design space of families of software systems in terms of patterns of structural organization, defining a *vocabulary* of component and connector types, as well as a set of *constraints* on how they can be combined. Styles help designers constrain design space exploration to within a set of legal structures to which the system must conform. However, while the structure of a system may be constrained by some style, there is still considerable design flexibility left for exploring the tradeoffs on many of the qualities that a system must achieve.

Definition 1 (*Architectural Style*)**.** Formally, we characterize an architectural style as a tuple $(\Sigma, \mathcal{C}_S)$, where:

- $\Sigma = (CompT, ConnT, \Pi, \Lambda)$ is an architectural signature, such that:
  - $CompT$ and $ConnT$ are disjoint sets of component and connector types. For conciseness, we define $ArchT \equiv CompT \cup ConnT$.
  - $\Pi : ArchT \rightarrow 2^{\mathcal{D}}$ is a function that assigns sets of symbols typed by datatypes in a fixed set $\mathcal{D}$ to architectural types $\kappa \in ArchT$. $\Pi(\kappa)$ captures properties associated with type κ. To refer to a property $p \in \Pi(\kappa)$, we simply write $\kappa.p$.
  - $\Lambda : ArchT \rightarrow 2^{\mathcal{P}} \cup 2^{\mathcal{R}}$ is a function that assigns a set of symbols typed by a fixed set $\mathcal{P}$ to components $\kappa \in CompT$. This function also assigns a set of symbols in a fixed set $\mathcal{R}$ to connectors $\kappa \in ConnT$. $\Lambda(\kappa)$ represents the ports of a component (conversely, the roles if κ is a connector), which define logical points of interaction with κ's environment. To denote a port/role $q \in \Lambda(\kappa)$, we write $\kappa :: q$.
- $\mathcal{C}_S$ is a set of structural constraints expressed in a constraint language based on first-order predicate logic in the style of Acme (Garlan et al., 2000) or OCL (Warmer and Kleppe, 2003) constraints (e.g., ∀ w:TASWorkflowT •∃ a:AlarmServiceT • connected(w,a) – "every TAS workflow must be connected to at least one alarm service").

In the remainder of this section, we assume a fixed universe $\mathcal{A}_\Sigma$ of architectural elements, i.e., a finite set of components and connectors for Σ typed by $ArchT$. The type of an architectural element $c \in \mathcal{A}_\Sigma$ is denoted by $type(c)$. We assume that elements of $\mathcal{A}_\Sigma$ are indexed and designate the ith element by $\mathcal{A}^i_\Sigma$.

A *configuration* is a graph that captures the topology of a legal structure of the system in a style $\mathcal{A}$ (we designate $\mathcal{A}$'s set of legal configurations by $\mathcal{G}^*_\mathcal{A}$).

Definition 2 (*Configuration*)**.** A configuration in a style $(\Sigma, \mathcal{C}_S)$, given a fixed universe of architectural elements $\mathcal{A}_\Sigma$, is a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ satisfying the constraints $\mathcal{C}_S$, where: $\mathcal{N}$ is a set of nodes, such that $\mathcal{N} \subseteq \mathcal{A}_\Sigma$, and $\mathcal{E}$ is a set of pairs typed by $\mathcal{P} \times \mathcal{R}$ that represent *attachments* between ports and roles.

To determine if two architectural elements are attached on any of their port/roles, we define the function $att : \mathcal{A}_\Sigma \times \mathcal{A}_\Sigma \rightarrow \mathbb{B}$ as $att(n, n') = \top$ if $\exists p \in \mathcal{P}, r \in \mathcal{R} \bullet n :: p \wedge n' :: r \wedge (p, r) \in \mathcal{E}$, and $att(n, n') = \bot$ otherwise. We say that two components are *bound* if there is a connector attached to any of their ports on both ends. This is captured by function $bnd : CompT \times CompT \rightarrow \mathbb{B}$, $bnd(n, n') = \top$ if $\exists\ n'' \in \mathcal{N}$, s.t. $att(n, n'') \wedge att(n'', n')$, and $bnd(n, n') = \bot$ otherwise.

3.2. Configuration data extraction

The first stage of our approach extracts the set of relevant attributes that correspond to different design decisions made to form any legal configuration (i.e., that conforms to the architectural style), which are provided as input to the process. Our approach is agnostic to the mechanisms employed to generate the set of configurations that conform to an architectural style:

this process is out of scope of this paper, but existing prior work has addressed this problem in a variety of ways (see Cámara et al. (2019) for one example).

The attributes extracted from a configuration $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ form a tuple of *design variable* values $\mathcal{D}_\mathcal{G}(C, T, P) \in \mathcal{DG}$, where:

- $C \in \mathbb{R}^n_{>0}$ is a vector that contains data items corresponding to constituent architectural elements of the configuration (e.g., the presence and number of specific components and connectors). Concretely, this vector is formed by concatenating the result of the following functions:
    1. *Architectural element presence extraction* $f_{ep} : \mathcal{G}^*_\mathcal{A} \to \{0, 1\}^{|\mathcal{A}_\Sigma|}$, returns a vector $\langle p_1, \dots, p_{|\mathcal{A}_\Sigma|}\rangle$ that encodes the presence of specific architectural elements (i.e., component and connector instances) in a configuration, where $p_i = 1$, $i \in \{1..|\mathcal{A}_\Sigma|\}$ if $\mathcal{A}^i_\Sigma \in \mathcal{N}$, and $p_i = 0$, otherwise.
    2. *Architectural type cardinality extraction* $f_{tc} : \mathcal{G}^*_\mathcal{A} \to \mathbb{N}^{|ArchT|}$, returns a vector $\langle x_{tc}(\kappa_1), \dots, x_{tc}(\kappa_{|ArchT|})\rangle$ encoding the number of component and connectors of each type present in a configuration. For $\kappa \in ArchT$, we define function $x_{tc} : ArchT \to \mathbb{N}$ as $x_{tc}(\kappa) = |\{n \in \mathcal{N} \mid type(n) = \kappa\}|$.
- $T \in \mathbb{R}^n_{>0}$ is a vector of data items that correspond to the topology of the configuration like the presence of certain attachments among architectural elements, and other topological measures like centrality indices, which characterize important nodes in the configuration topology (Bonacich, 1987; Borgatti, 2005). Concretely, this vector is formed by concatenating the results of the following functions:
    1. *Binding presence extraction* $f_{bp} : \mathcal{G}^*_\mathcal{A} \to \{0, 1\}^{|\mathcal{A}_\Sigma| \cdot |\mathcal{A}_\Sigma|}$ returns a vector $\langle p_{1,1}, \dots, p_{|\mathcal{A}_\Sigma|,1}, \dots, p_{|\mathcal{A}_\Sigma|,|\mathcal{A}_\Sigma|}\rangle$ that encodes the presence of bindings between specific components, with $p_{i,j} = 1$, $i, j \in \{1..|\mathcal{A}_\Sigma|\}$ if $bnd(\mathcal{A}^i_\Sigma, \mathcal{A}^j_\Sigma)$, and $p_{i,j} = 0$ otherwise.
    2. *Binding type cardinality extraction* $f_{btc} : \mathcal{G}^*_\mathcal{A} \to \mathbb{N}^{|CompT| \cdot |CompT|}$, returns a vector $\langle x_{btc}(\kappa_1, \kappa_1), \dots, x_{btc}(\kappa_{|CompT|,1}), \dots, x_{btc}(\kappa_{|CompT|,|CompT|})\rangle$ encoding the number of bindings between specific pairs of component types. For the pair of component types (κ, κ'), we define function $x_{btc} : CompT \times CompT \to \mathbb{N}$ as $x_{btc}(\kappa, \kappa') = |\{(n, n') \in \mathcal{N} \times \mathcal{N} \mid type(n) = \kappa \wedge type(n') = \kappa' \wedge bnd(n, n')\}|$.
- $P \in \mathbb{R}^n$ is a vector containing data items corresponding to the values of relevant configuration parameters. We assume that these can be directly obtained from the values of properties associated with the different architectural elements of the configuration (e.g., the configuration parameter for the number of maximum service retries in TAS is stored in property TASWorkflow0.max_timeouts, where TASWorkflow0 is an instance of TASWorkflowT).

3.3. Data aggregation and normalization

The second input to our approach is a set of vectors $\mathcal{RG}$ of the form $\mathcal{R}_\mathcal{G} = \langle r_1, \dots, r_n\rangle$, $r_i \in \mathbb{R}, i \in \{1..n\}$ containing *response variables* that correspond to the values of the quantified metrics for the different quality dimensions in a configuration $\mathcal{G}$. Our technique is agnostic to the mechanisms employed to quantify the quality metrics of a configuration. However, in the particular instantiation of the approach used in this paper, we obtain these values by checking a variety of probabilistic temporal logic properties encoded in an extension of PCTL using HaiQ (Cámara, 2020), a tool that performs probabilistic model checking on collections of structural design variants that uses Alloy (Jackson, 2002) and PRISM (Kwiatkowska et al., 2011) in its backend.

The purpose of data aggregation and normalization is to generate a *correlations table* that can be provided as input to PCA:

- *Data aggregation*. Given a design variable value tuple $\mathcal{D}_\mathcal{G}(C, T, P) \in \mathcal{DG}$, and a response variable vector $\mathcal{R}_\mathcal{G} = \langle r_1, \dots, r_n\rangle \in \mathcal{RG}$ for the same configuration $\mathcal{G}$, we define the *configuration sample* for $\mathcal{G}$ as $\mathcal{R}_\mathcal{G} \frown \mathcal{D}_\mathcal{G}$, where $\frown$ denotes concatenation. The (non-normalized) correlations table is formed by the samples that correspond to all input configurations.
- *Data normalization*. The correlations table contains variables that span varying degrees of magnitude and range. To avoid bias in PCA towards variables that may have a higher magnitude, we scale the data employing unity-based normalization, meaning that for any data item in the correlations table $x_{i,j}$ for sample i and variable j, the new value of the data item is defined as $x'_{i,j} = (x_{i,j} - x^{min}_j)/(x^{max}_j - x^{min}_j)$, where x^{min}_j, x^{max}_j are the minimum/maximum values of j across all samples.

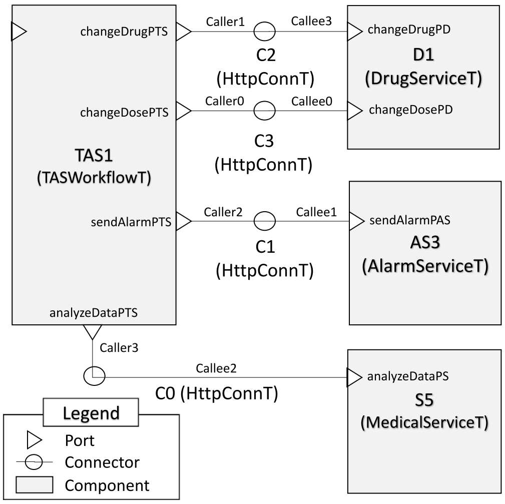
Example 1. Fig. 4 shows a TAS configuration and an excerpt of its encoding in the correlations table. The first and second topmost tables show the presence of architectural elements and type cardinalities (f_{ep} and f_{tc}, respectively). In this case, we can observe that the cardinality of all architectural types is 1, except for HttpConnT type, of which there are four instances. The table at the bottom describes the presence of bindings between components (f_{bp}).

3.4. Principal component analysis

Data resulting from analyzing architectural spaces usually contain a large amount of information, which is often too complex to be easily interpreted. Principal Component Analysis (PCA) (Jolliffe, 1986) is a statistical projection method commonly used in ML and natural science that can facilitate understanding that information. To begin with, PCA can help to find out in what respect some architectural configurations differ from others, and which variables contribute to this difference. In some cases, variables contribute in the same way (i.e., are correlated) or independently. Moreover, PCA also enables quantifying the amount of useful information in a data set, as opposed to noise or meaningless variations.

If we consider the data in the correlations table geometrically, we can say that two samples (i.e., architectural configurations) are *similar* if they have close values for most variables (i.e., they are in the same region of the multidimensional space) and *different*, otherwise. Considering this, the purpose of PCA is finding the directions in space in which the distance between points is the largest. That is equivalent to finding the linear combinations of the variables that contribute most to making the samples (i.e., configurations) different from each other. These directions or linear combinations are called *principal components*.

Principal components (PC) are computed in an iterative manner, in such a way that the first PC is the one that carries the most information (most explained variance), whereas the second PC will carry the maximum share of the information not taken into account by the previous PC, and so on. All PCs are orthogonal to each other and each one carries more information than the next one. In fact, this is one of the characteristics of PCA that makes it appealing as an underlying mechanism to enable the explainability of architectural design tradeoff spaces: the interpretation of PCs can be prioritized, since the first PCs are known to carry the most information. Indeed, it is often the case that only the first two PCs contain genuine information, whereas the rest describe noise (Jolliffe, 1986).



f_{ep}	TAS1	1	AS1	0
	D1	1	AS2	0
	S1	1	AS3	1
	S2	0	C0	1
	...	...	...	...
	S5	0	C3	1

f_{tc}	TASWorkflowT	1	AlarmServiceT	1
	DrugServiceT	1	HttpConnT	4
	MedicalServiceT	1		

f_{bp}	TAS1-TAS1	0	TAS1-AS1	0
	TAS1-D1	1	AS2-S1	0
	TAS1-S1	0	AS3-TAS1	1
	TAS1-S5	1	AS1-D1	0
	...	...	...	...
	S1-S2	0	AS3-AS3	0

Fig. 4. Sample TAS configuration (left), along with an excerpt of its encoding (right).

The main results of PCA consist of three complementary sets of attributes: (i) *variances*, which tell us how much information is taken into account by the successive PCs, (ii) *loadings*, which describe relationships between variables, and (iii) *scores*, which describe properties of the samples. In this paper, we focus on variances and loadings, which will tell us what are the main variables (i.e., either design or response variables) that contribute the most (and in what way) to the differences among configurations.

Example 2. The PCA loadings plot of the samples analyzed for TAS is shown in Fig. 5. The plot contains two ellipses that indicate how much variance is taken into account. The outer ellipse is the unit-circle and indicates 100% of the explained variance, whereas the inner ellipse indicates 50% of the explained variance. Variables that are found between the edges of the two ellipses, and particularly those positioned near the edge of the outer ellipse, are those that are more important in differentiating the configurations. This plot shows that the first two PCs carry a large amount of information, explaining 76% of the variance of data, with PC1 explaining the most (71%) and PC2 explaining much less variance (5%).

QoS metrics like reliability, cost, and response time are all important to differentiate configurations with response time being the most relevant (close to 1 in PC1, which accounts for more than 70% of the overall variability). Reliability and cost (upper-left quadrant) are also important for PC1, placed below −0.5 in PC1, but comparatively have less influence than response time in overall variation.

In addition to teasing out the most important variables, the plot displays the relationships between variables. In the plot, the angle between the vectors that go from the origin of coordinates to a variable point is an approximation of the correlation between the variables. A small angle indicates that the variables are positively correlated, an angle of 90 degrees indicates the variables are not correlated, and an angle close to 180 degrees indicates the variables are negatively correlated. In our example, we can observe that reliability and cost are positively correlated, whereas response time is negatively correlated with both of them. These observations are consistent with the results in Fig. 2, which show that low response times and high reliability correspond to higher costs.

So far, we have been discussing QoS variables, but the loading plot also enables architects to observe the influence of design variables on variability. Here, we can see in the upper-right quadrant of the ellipse that some of the most influential variables for PC1 correspond to the presence of alarm service instance AS3 in a configuration, as well as to its binding to the workflow (TASWorkflow0). We observe that all the design variables related to AS3 are positively correlated with reliability and cost, and negatively correlated with response time. This indicates that the selection of AS3 has a remarkable influence on the qualities of the resulting configurations and is consistent with the fact that the alternative alarm service implementations have considerably higher failure rates and response times than AS3, as well as lower cost per invocation (see Fig. 1a). Also, the alarm service is invoked more times in the workflow than any other service. Consequently other services like MS5, which are also influential and have the same QoS correlations as AS3, have a comparatively moderate impact (its associated design variables are within the inner edge of the ellipse) because they are invoked only once in the workflow. In the bottom-right quadrant, we have the symmetric case, with variables associated with the presence of AS2, which in contrast with AS3, are positively correlated with response time, and negatively correlated with cost and reliability (most importantly along the PC1 horizontal axis. This is also consistent with the data in the table of Fig. 1, which shows that AS3 has the highest failure rate and lowest cost. Regarding configuration parameters, we can see that, as expected, timeout length for service invocations is positively correlated with response time with respect to PC1 and negatively correlated to reliability and cost. In contrast, and although less influential, the maximum number of retries for service invocations is positively correlated with reliability and cost. This observation is consistent with the fact that more service invocation retries lead to increased reliability and cost.

3.5. Decision tree learning

Decision tree learning is a supervised learning technique used in statistics, data mining, and machine learning that allows the prediction of a target variable's value based on other variables' values (Breiman et al., 2017). It is a supervised technique that can be used to grow classification trees (to predict categorical values) and regression trees (to predict numerical values). We decided to apply decision tree learning as part of ExTrA, as it is particularly useful in contexts that involve complex datasets with high dimensionality and heterogeneous data types (Breiman et al., 2017). Decision trees are generated based on *recursive binary partitioning*, a method relying on repeatedly computing partitions of the data that minimize the residual sum of squares (for regression trees) or the Gini index (for classification trees) (Breiman et al.,

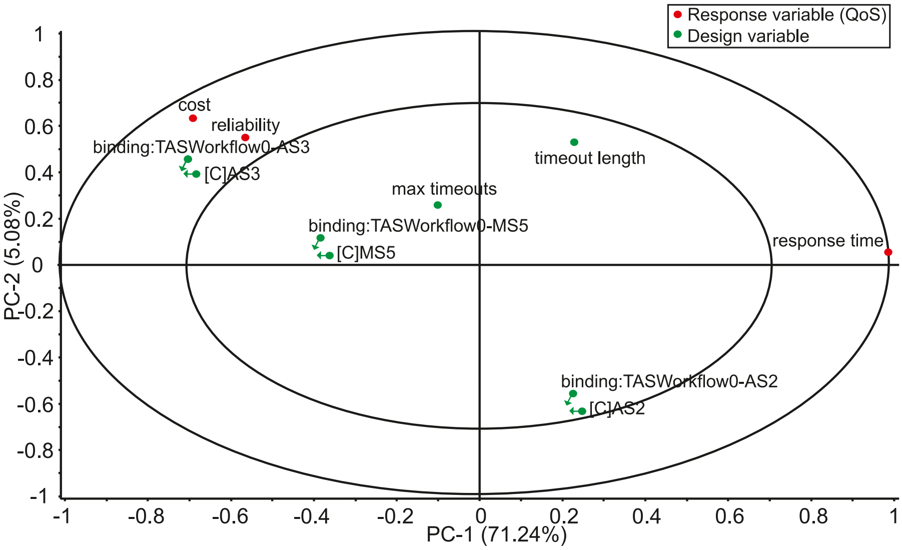


Fig. 5. Correlation loading PCA plot for TAS.

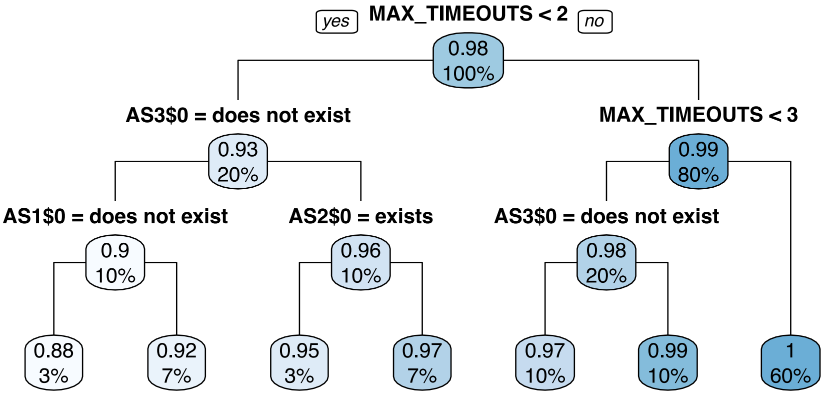


Fig. 6. Decision tree plot for the Tele-Assistance System V1, predicting reliability.

2017). As a subsequent step, in order to minimize overfitting, the tree is pruned by computing the subtree that minimizes the mean squared prediction error.

Decision tree learning requires selecting a target variable whose value should be predicted or classified. We leverage the insights from analyzing PCA plots to inform the target variable selection. Variables that are of relevance to explain the variance in the data are good candidates as target variables. In our approach, we are mainly interested in the impact of architectural design decisions on quality attributes. For instance, by looking at Fig. 2, it can be seen that there are several levels of reliability with several thresholds indicating a "stepwise transition" between them. Decision trees can be used to explain what these steps entail (e.g., with respect to the selection of input parameters or architectural design decisions). Analyzing such decision trees can in turn help architects to better understand tradeoffs and make decisions more deliberately.

Fig. 6 shows a decision tree plot for TAS, predicting the level of reliability depending on other numerical variables. The tree's root indicates the most relevant condition (i.e., `MAX_TIMEOUTS` < 2). If the condition is fulfilled, the left branch of the tree should be considered. If `MAX_TIMEOUTS` is larger or equal to 2, the right branch is considered.

In the example, it can be seen that for low maximum numbers of timeouts, a rather low degree of reliability is achieved. The existence of component AS3 is relevant as well: if it is included in a configuration, reliability is higher than if it is not. Apart from that, the inclusion of AS1 and AS2 are important as well. If AS1 exists, reliability is higher and if AS2 exists, reliability is slightly lower. It should be noted that although `MAX_TIMEOUTS` is not a relevant variable to explain the variance of configurations in the dataset, it is a relevant variable as a predictor for reliability. Insights like this can help with the fine-grained analysis of the dataset and specific quality attributes.

4. Demonstration

The objective of our demonstration is to: (i) assess the *feasibility* of linking design decisions to requirement satisfaction (RQ1), (ii) quantify the impact of architectural design decisions on the different qualities across the architectural design space, and (iii) assess the tradeoff between the information reduction and the amount of variance explained (RQ2).

In this section, we first describe our experimental setup. We then briefly introduce a scenario that we have incorporated into our demonstration in addition to TAS. Finally, we discuss results, relating them to our research questions.

4.1. Experimental setup

We generated the set of architectural configurations for the different scenarios included in our study and their QoS metrics

Table 1
Dataset dimensions for our experiments.

Case study	# Variables							#samples
	QoS	f_{ep}	f_{tc}	f_{bp}	f_{btc}	Parameters	Total	
Tele-Assistance System V1	3	10	10	27	27	2	79	3750
Tele-Assistance System V2	3	10	10	27	27	2	79	3750
Network Architecture V1	5	9	4	46	11	3	78	60000
Network Architecture V2	5	9	4	70	15	3	106	60000
Network Architecture V3	5	9	4	40	7	3	68	60000

using an extended version of HaiQ that implements the data extraction, aggregation, and normalization procedures described in Sections 3.2–3.3 and the set of models for TAS and the network virus example described in Cámara (2020).

Table 1 describes the number of variables and samples included in the datasets generated for all the variants of the case studies.

Data analysis (i.e., Principal Component Analysis and Decision Tree Learning) was performed using R.[1] For PCA, we used the prcomp[2] package and for decision tree learning, rpart was used. Besides following the normalization steps described below, we centered the variables in PCA, shifting all variables to be zero centered. Centering the data subtracts the mean of each column from the values in that column. Given that we are interested in understanding the variance in a dataset, centering the variables is beneficial as it ensures that the first principal component explains the direction of maximum variance. For decision tree learning with rpart,[3] we used the default parameters.

4.2. Scenario: Tele-assistance system

In Example 2 we described the correlation loadings plot for the Tele-Assistance System, illustrating the main variables that influence configuration variability, as well as their correlations. To further demonstrate the consistency of our approach in providing links between architectural design decisions and the quality of configurations, in this section, we also consider a variant of TAS (V2) that has been altered by modifying the reliabilities and response times of AS3, which become worse (failure rate = 0.4, response time = 7, and cost = 6.8), and of MS5, which improve (failure rate = 0.05, response time = 10, cost = 11.9).

Results: Principal Component Analysis

Fig. 7 shows the correlation loading PCA plot for TAS V2. In the figure, we can observe that multiple variables are in different positions, compared to the original version of TAS (cf. Fig. 5). If we start by observing QoS variables, we can notice that the only exception is response time, which remains exactly in the same position as in TAS V1. In contrast, we can observe how both cost and reliability have lost influence in terms of explaining variability, moving closer to the origin of the horizontal PC1 axis. This indicates that while response time is still a major contributor to configuration QoS variation, the other two variables contain less variability, probably because one of the main factors that influenced obtaining configurations with high reliability and cost was the inclusion of AS3, which in TAS V2 is degraded to have a higher failure rate. Hence, since no alarm service is particularly reliable in TAS V2, there is a reduction in the variability along that dimension. Indeed, moving on to design variables, we can observe that the variables associated with the inclusion of AS3 in a configuration have moved from the top-left to the top-right quadrant of the plot. This indicates that the inclusion of AS3 is much less relevant now in terms of explaining variability, being much closer to the origin of the horizontal PC1 axis. Beyond that, we can also observe that its correlations have also changed. In particular, these variables are now negatively correlated with cost, and even more with reliability, and positively correlated with response time. These results are consistent with the fact that both the reliability and the response time of AS3 have now been degraded.

Interestingly, we can also observe that the variables associated with the inclusion of AS2 have moved from the bottom-right to the bottom-left quadrant of the plot. Although the influence on variability explanation remains relatively low, the correlations of these variables have now been inverted, being positively correlated with cost and reliability along the PC1 axis, and negatively with response time. This is consistent with the fact that relative to the quality attributes of the other components, the failure rate of AS2 is now on the high end of the spectrum.

Other variables in the plot have also been displaced with respect to TAS V1, but their changes are not that significant, given that their movement along the horizontal PC1 axis is relatively small, and movements along the PC2 vertical axis are not that representative, even if they are noticeable, due to the small amount of explained variance of PC2, relative to PC1 (approximately 7% for PC2 vs. 69% for PC1).

Results: Decision Tree Learning

Fig. 8 shows a decision tree plot for TAS V2, predicting the level of reliability depending on other numerical variables. The tree's root indicates the most relevant condition (i.e., `MAX_TIMEOUTS` < 2). It can be seen that for a low maximum numbers of timeouts, a rather low degree of reliability is achieved. The inclusion of component AS3 is relevant as well: if it is included, reliability is lower than if it not included. Apart from that, the inclusion of AS1 is of importance as well. If AS1 is included, reliability is higher. The insight that AS3 leads to lower reliability is in line with our observations from the PCA plot: given that the reliability of AS3 has been degraded in V2, including this component leads to an overall lower reliability level.

4.3. Scenario: Network architecture

Architecting network-based systems that are resilient to uncontrollable environment conditions, such as network delays, or undesirable events such at virus infections, entails structuring the system in a way that maximizes the chances of continued service provision in spite of the adverse conditions that it is subject to. The scenario introduced by Kwiatkowska et al. (Kwiatkowska et al., 2009) models the progression of a virus infecting a network formed by a grid of N×N nodes. The virus remains at a node that is infected and repeatedly tries to infect any uninfected neighbors by first attacking the neighbor's firewall and, if successful, trying to infect the node. In the network there are 'low' and 'high' nodes divided by 'barrier' nodes that scan the traffic between them and have better chances of detecting any potential virus infection attempts than low and high nodes. Ideally, the architecture of the

[1] Our R scripts and datasets are publicly available at: github.com/cmu-able/exTrA-material

[2] https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/prcomp

[3] https://www.rdocumentation.org/packages/rpart/versions/4.1.16/topics/rpart

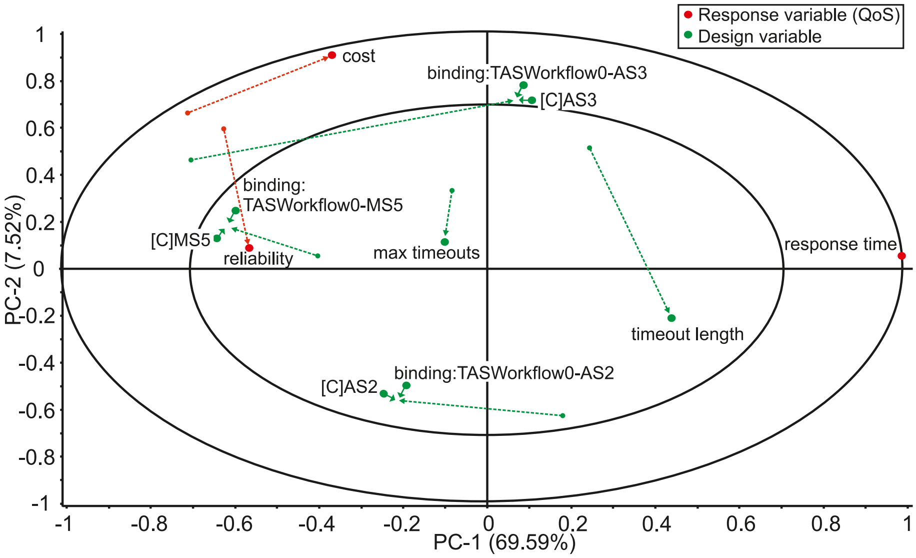


Fig. 7. Correlation loading PCA plot for TAS V2. Dashed arrows represent variable displacements with respect to TAS V1.

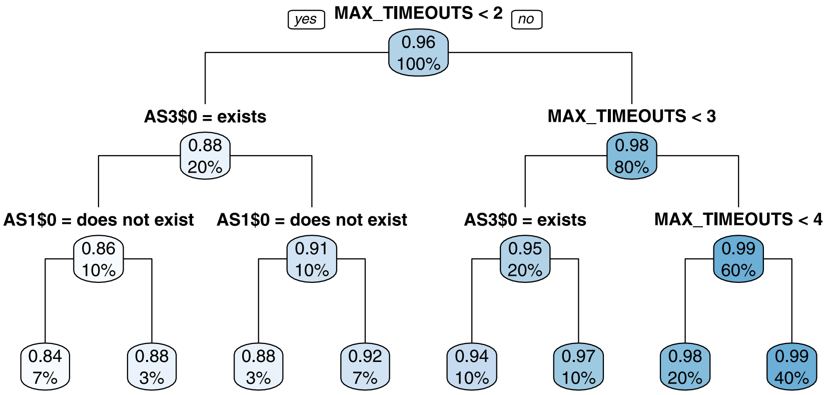


Fig. 8. Decision tree plot for the Tele-Assistance System V2, predicting reliability.

network should: (i) minimize the probability of successful infection of high nodes in the network within some time bound, and (ii) maximize the number of node infection or attack attempts that the virus carries out to spread itself through the high nodes. Initially, only one corner 'low' node is infected.

We carried out the analysis of three variants of this system:

1. A variant in which only the communication between low nodes and high nodes has to go through barrier nodes (as originally described in Kwiatkowska et al. (2009).
2. An unconstrained variant in which there is no enforcement of communication between high nodes and low nodes through barrier nodes.
3. A restrictive variant that enforces communication between all nodes through a barrier node.

Results: Principal Component Analysis

(Fig. 9) displays the first two PCs of each variant, which explain approximately 99% of the data variation ($\geq$ 98% for PC1 and $\leq$ 1% for PC2). This shows a clear dominance of PC1, which explains most of the data variation, with a marginal contribution of PC2.

Moreover, in all cases, we can observe that both QoS metrics for the virus infection success (for the different time bounds of 50, 100, and 150 time units), and the maximum number of virus attacks are very important to differentiate configurations.

Variant 1. We can observe that both QoS metrics for the virus infection success probability and maximum number of virus attacks are at opposite ends of the horizontal axis and are negatively correlated. This indicates that higher probability of infection success requires in principle fewer virus infection attempts. Although this can be counter-intuitive, it can be explained by the fact that the values for the virus attack success probability variable are obtained from a time-bounded probabilistic analysis of the network model, meaning that scenarios in which the virus successfully infects high nodes after {50, 100, 150} time units are not captured in the samples. In contrast, the values for the maximum number of attacks are not time-bounded.

Concerning design variables, we can observe that the most influential are the probability of individual infection when the virus is attacking a node (infect), followed by the number of bindings between high nodes ([C]binding:highNode-highNode). In both cases, these variables are positively correlated with the probability of having all high nodes infected within some time bound: this makes sense, given that higher values of infect mean more effective infection attempts, and a higher number of bindings between high nodes represent more opportunities for the virus to spread in fewer 'hops' between high nodes, once the barrier nodes have been breached.

Variant 2. We can observe that infect is still the most influential variable to explain data variation. In addition to that, the variable has moved even closer to the variables for the overall

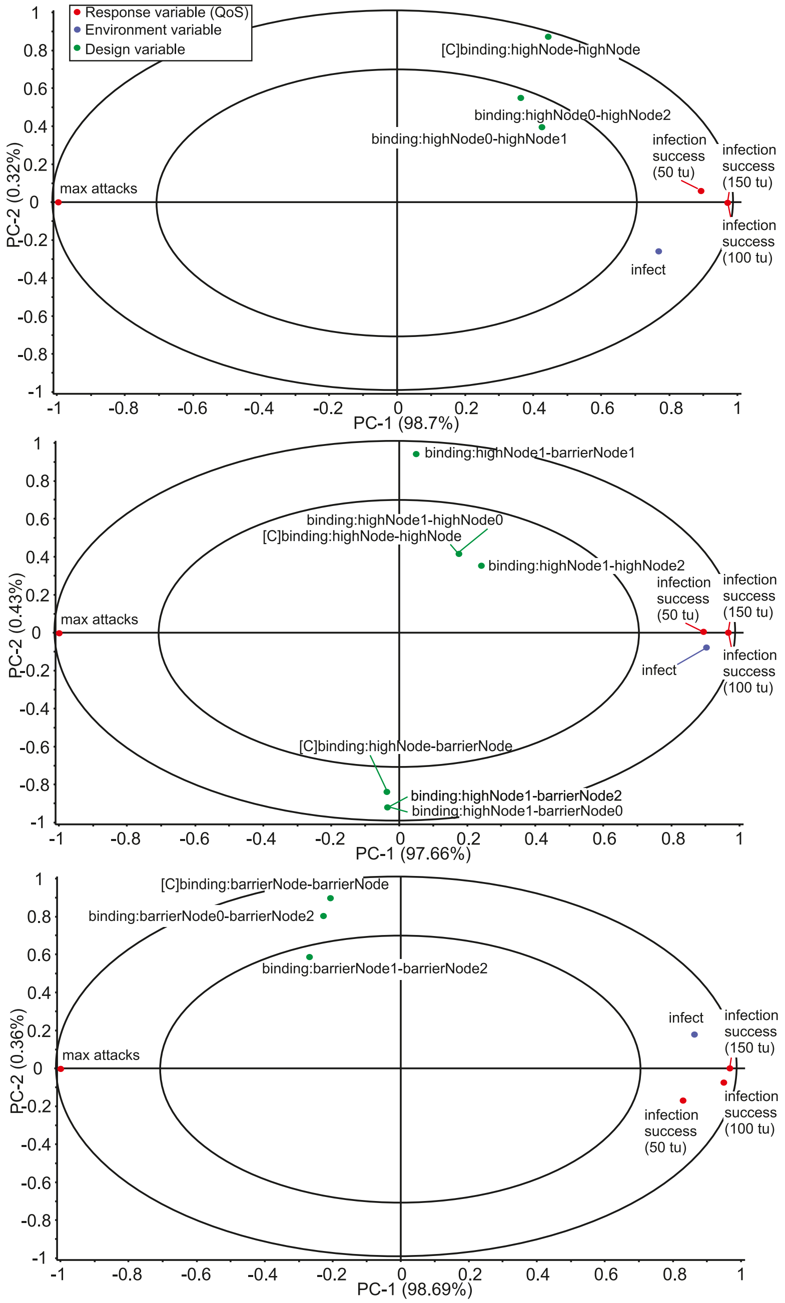


Fig. 9. Correlation loading PCA plots for the network architecture: (top) communication scanned between through high nodes and barrier nodes, (middle) no communication scan enforced among nodes, (bottom) communication scan between all nodes enforced.

probability of infection, meaning that the correlation is stronger now. This is consistent with the fact that in this variant, communication through barrier nodes is not enforced, so it makes sense that a higher probability of node infection under attack has more influence over the overall probability of high node infection. If we take a look at the inner ellipse of the PCA plot, we can observe that the number of bindings between high nodes is no longer an influential variable, given that the lack of enforcement of communications through barrier nodes dilutes its influence on the spread of the virus.

If we focus on the bottom and top-center parts of the plot, we can observe that in this case the influence on variability in terms of bindings is slightly more slanted towards bindings between barrier and high nodes. However, this influence can also be considered as marginal, given that these variables are influential only in the context of PC2, which explains less than 1% of the variation. If we consider all the observations together, they are consistent with the fact that topology does not have much influence on the resilience of the network when communications through more secure nodes is never enforced.

Variant 3. In this variant, it can be observed that the probability of infection is still the most influential variable, but is not as strongly correlated with the overall probability of high node infection as in variant 2. We can observe that the variables associated with the bindings between barrier nodes (e.g., ([C]binding:barrierNode-barrierNode) also have a moderate influence on variability, although in this case they are negatively correlated with the probability of infection of high nodes. This makes sense from the perspective that more bindings among barrier nodes represent more opportunities to scan the traffic

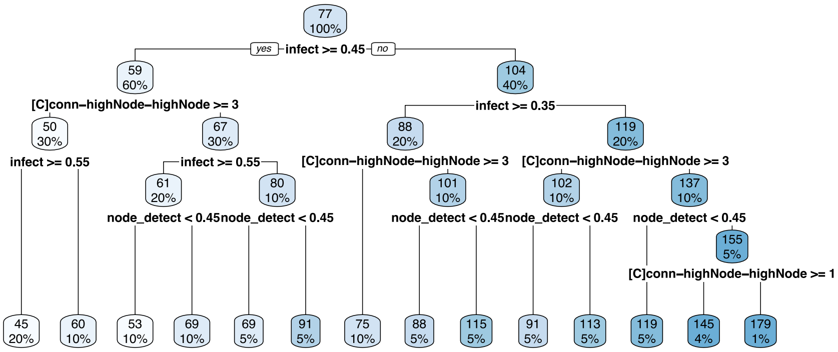


Fig. 10. Decision tree for the first variant of the network architecture system (predicting the expected maximum number of attacks).

before it arrives at high nodes. Given that in this variant all communication between low and high nodes has to go through a barrier node, the fact that, on average, more 'hops' through barrier nodes are required represents more opportunities to thwart or at least delay node infections, compared to topologies that include fewer bindings among barrier nodes.

Results: Decision Tree Learning

Fig. 10 shows a decision tree plot for the first variant of the network architecture system, predicting the expected maximum number of attacks needed for a highly infected system. The tree's root indicates that the most relevant condition is the infection rate (infect). For a high rate of infection, the expected maximum number of attacks is low. The number of connections between high nodes is relevant as well: if it is high, fewer attacks are needed. Also the probability of node detection is important: if it is low, the expected number of attacks is lower than if it is high. These insights confirm our observations from analyzing the PCA plot. The infect variable and the number of bindings between high nodes are negatively correlated with the max attacks variable. Stakeholders analyzing PCA plots can use these observations to generate decision trees and analyze the concrete thresholds of variable values leading to different outcomes (in this case, in terms of max attacks).

4.4. Discussion

We have shown the applicability of our approach by demonstrating it in two cases. In practice, however, it is not trivial to construct design spaces and collect quantitative data on requirements satisfaction. The creation of the dataframes would require ways to generate architectural configurations, as well as a simulator, a digital twin, or real-world data, to collect realistic quality attribute measurements. Taking these requirements into account, we believe that these ideas are applicable (with adaptations) to systematic management of variability techniques that are currently used in industry, such as product line architectures.

(RQ1) Feasibility — Linking architectural design decisions with requirement satisfaction. PCA analysis results performed on the various TAS and network architecture scenarios have shown that our approach is able to extract information that links QoS variables and design variables, providing a basis for explaining what design variables are the best candidates to describe important QoS variation. When studying the relation among QoS variables, results obtained across the various scenarios considered in our demonstration are consistent with observations obtained from existing analysis techniques (Cámara, 2020; Cámara et al., 2019). For the relation between design variables and QoS variables, results are also consistent with observations obtained from careful examination of models and simulations of the systems we have studied. Moreover, our results have been obtained from two different types of architecture (a centralized service-based system and a decentralized network architecture). In the centralized system, component variability has a more prominent role in explaining QoS variation, whereas in the decentralized system, configuration topology explains most of the QoS variation. The ability of the approach to yield insightful results in both cases indicates its potential applicability to a broad range of scenarios.

(RQ2) Feasibility — Quantifying impact of architectural design decisions on qualities. The application of decision tree learning has enabled us to extract informative information with respect to how qualities are affected by design decisions. Concretely, the generated decision trees indicate thresholds in variable values that lead to differences in the fulfillment of QoS variables. For the Tele-Assistance System, for instance, the generated trees show the importance of a threshold of 2 for the maximum number of timeouts, whereas for the network architecture system, the infection success rate and the number of connections between high nodes are identified as relevant. These results cannot be obtained by simply studying PCA plots and hence have the potential to facilitate the work of an architect by making the relations between variables explicit. As a consequence of studying these trees, parameters and design decisions can be chosen more deliberately. Moreover, our observations confirm the insights we obtained by studying PCA plots and existing analysis techniques.

We found that both PCA and decision tree learning were applicable to all versions of the systems and datasets that we used in this paper. Other techniques used to understand large datasets in machine learning are Multiple Correspondence Analysis (MCA) and clustering algorithms. We investigated these approaches as well. MCA is most beneficial to explain variance in datasets with categorical variables and did not yield interesting results for the datasets used in this paper. In contrast to other system properties, architectural configurations and quality attributes can be easily captured in quantitative variables, which is why PCA was a more appropriate technique in these cases. Clustering appears to be a promising area to explain the natures of different categories of configurations. However, in our example systems, the identified clusters were difficult to understand and mainly reflected the tradeoffs we identified based on PCA (e.g., resulting in two resulting clusters, with one cluster being cost-effective but having low reliability and high response times, and the other cluster having opposite characteristics). There may be circumstances in which these additional techniques could provide useful insight, but in this paper we have not elaborated on them because of their limited usefulness for the systems studied.

(RQ3) Information Reduction-Explained Variance Tradeoff.

Table 2 summarizes the information reduction and explained variance for the two scenarios described in the paper. In the table, information reduction is calculated as the percentage of the original variables in the dataset that remain as relevant in

Table 2
Information reduction and explained variance summary.

Case study	#dataset vars.	#relevant PCA vars.	Information reduction	Explained variance	Residual variance
Tele-Assistance System V1	79	42	46.84 %	76.32 %	23.68 %
Tele-Assistance System V2	79	20	74.68%	73.11%	26.89%
Network Architecture V1	78	12	84.62%	99.02%	0.98%
Network Architecture V2	106	26	75.47%	98.09%	1.91%
Network Architecture V3	68	16	76.47%	95.32%	4.68%
Average	82	23.2	71.62%	88.37%	11.63%

the PC1–PC2 correlation loadings plot (i.e., positioned within the 50%–100% explained variance ellipses), whereas the total explained variance for PC1–PC2 is one of the outputs provided by PCA. Total residual variance corresponds to the remainder of PCs, i.e., variance that is left unexplained by PC1 and PC2.

We can observe that in all scenarios, there is a remarkable reduction in the information that has to be examined by an architect to analyze the tradeoff space, which is in the range 46%–84% of the overall number of variables available in the dataset. At the same time, the total residual variance ranges between 0.98 and 26.89%. Although non-negligible, these are moderate levels of residual variance, especially if we consider them in the context of the drastic dimensionality reduction in the set of explanatory variables.

5. Related work

Evaluation of software architectures under uncertainty is a subject that has been broadly explored (Sobhy et al., 2021). Due to space constraints, we focus on the subset of works akin to our proposal, which can be categorized into:

Architecture-based quantitative analysis and optimization approaches, which focus on analyzing and optimizing quantitative aspects of architectures using mechanisms that include e.g., stochastic search and Pareto analysis (Aleti et al., 2009; Bondarev et al., 2007; Martens et al., 2010). Other recent approaches to architectural design synthesis and quantitative verification (Calinescu et al., 2017; Cámara, 2020; Cámara et al., 2019) generate and analyze alternative system designs, enabling exploration of quantitative guarantees across the design space. These techniques ((Cámara, 2020; Cámara et al., 2019) being our prior work) do not address explainability, but produce (large) datasets that can be used as input to the approach described in this paper.

Learning-based architecture evaluation adopts ML techniques to enhance the evaluation with observations of system properties over time (Calinescu et al., 2011; Cámara et al., 2020; Esfahani et al., 2013; Sobhy et al., 2020). These works employ Bayesian learning (Calinescu et al., 2011) to update model parameters, Model Tree Learning (MTL) to tune system adaptation logic (Esfahani et al., 2013), and reinforcement learning (Sobhy et al., 2020; Cámara et al., 2020) to analyze architectural decisions made at run-time.

While all the approaches described above provide some form of architectural tradeoff analysis (sometimes employing ML techniques), none of them makes any claims about explicitly linking design variables with requirements satisfaction or facilitating the explainability of the design tradeoff space. Indeed, a recent comprehensive literature review on architectural evaluation under uncertainty (Sobhy et al., 2021) reveals no approaches covering the research gap addressed by our technique.

6. Conclusions and future work

In this paper, we have presented what is, to the best of our knowledge, the first approach that explicitly relates QoS and architectural design variables using dimensionality reduction techniques employed in ML and other sciences, enabling architects to interpret the main tradeoffs of an architectural design space based on a graphical summary of the relations among the main variables that explain differences between configurations, as well as on decision trees that illustrate the impact of concrete design decisions on system qualities. Our results illustrate the feasibility of the approach both in terms of identifying the main sources of variability and design/QoS variable correlations (RQ1), as well as of quantifying the impact of design decisions on QoS dimensions (RQ2). Moreover, the results across all five scenarios included in our study indicate that a remarkable reduction in the amount of information required to explain the main tradeoffs of an architectural design space is attainable while the reduction in explained variance remains moderate (RQ3).

Although our approach works well in the case studies presented, PCA works optimally primarily in situations where variable correlations are linear, or an approximation thereof. Future work will involve exploring alternatives to PCA that enable the analysis of systems with strong non-linear correlations. Moreover, our approach is currently limited to component-and-connector static architectures with binary connectors. Our future work will also explore extensions to the catalogue of metrics and extraction functions required to enable richer analysis of various styles of architectural representation, including dynamic architectures, as well as the development of tools to help users easily interpret these analysis results.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Javier Cmara reports financial support was provided by FEDER/Ministerio de Ciencia e Innovacin. David Garlan reports financial support was provided by the Office of Naval Research. David Garlan reports financial support was provided by the NSA. Rebekka Wohlrab reports financial support was provided by the Knut and Alice Wallenberg Foundation.

Data availability

We have shared a link to a repository that contains data and scripts in our manuscript.

Acknowledgments

This work was partially supported by the Spanish Government (FEDER/Ministerio de Ciencia e Innovación – Agencia Estatal de Investigación) under project COSCA (PGC2018-094905-B-I00), by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, Sweden, by award N00014172899 from the Office of Naval Research, United States of America, and the NSA, United States of America under Award No. H9823018D0008. Any views, opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views the funding agencies.

References

Aleti, A., Bjornander, S., Grunske, L., Meedeniya, I., 2009. ArcheOpterix: An extendable tool for architecture optimization of AADL models. In: ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software. pp. 61–71.

Bonacich, P., 1987. Power and centrality: A family of measures. Am. J. Sociol. 92 (5), 1170–1182.

Bondarev, E., Chaudron, M.R.V., de Kock, E.A., 2007. Exploring performance trade-offs of a JPEG decoder using the deepcompass framework. In: 6th WS on Software and Performance. In: WOSP, ACM, pp. 153–163.

Borgatti, S.P., 2005. Centrality and network flow. Social Networks 27 (1), 55–71.

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

Calinescu, R., Ceska, M., Gerasimou, S., Kwiatkowska, M., Paoletti, N., 2017. Designing robust software systems through parametric Markov chain synthesis. In: International Conference on Software Architecture. ICSA, IEEE, pp. 131–140.

Calinescu, R., Ceska, M., Gerasimou, S., Kwiatkowska, M., Paoletti, N., 2018. Efficient synthesis of robust models for stochastic systems. J. Syst. Softw. 143, 140–158.

Calinescu, R., Johnson, K., Rafiq, Y., 2011. Using observation ageing to improve Markovian model learning in QoS engineering. In: 2nd ACM/SPEC International Conference on Performance Engineering. ICPE '11, ACM, pp. 505–510.

Cámara, J., 2020. HaiQ: Synthesis of software design spaces with structural and probabilistic guarantees. In: FormaliSE@ICSE 2020: 8th International Conference on Formal Methods in Software Engineering. ACM, pp. 22–33.

Cámara, J., Garlan, D., Schmerl, B., 2019. Synthesizing tradeoff spaces with quantitative guarantees for families of software systems. J. Syst. Softw. 152, 33–49.

Cámara, J., Muccini, H., Vaidhyanathan, K., 2020. Quantitative verification-aided machine learning: A tandem approach for architecting self-adaptive IoT systems. In: International Conference on Software Architecture. ICSA, IEEE, pp. 11–22.

Cámara, J., Silva, M., Garlan, D., Schmerl, B.R., 2021. Explaining architectural design tradeoff spaces: A machine learning approach. In: Biffl, S., Navarro, E., Löwe, W., Sirjani, M., Mirandola, R., Weyns, D. (Eds.), Software Architecture - 15th European Conference, ECSA 2021, Virtual Event, Sweden, September 13-17, 2021, Proceedings. In: Lecture Notes in Computer Science, vol. 12857, Springer, pp. 49–65. http://dx.doi.org/10.1007/978-3-030-86044-8_4.

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.

Garlan, D., 2010. Software engineering in an uncertain world. In: Proc. of the Workshop on Future of Software Engineering Research. FoSER, pp. 125–128.

Garlan, D., Monroe, R.T., Wile, D., 2000. Acme: Architectural description of component-based systems. Found. Compon.-Based Syst. 68. 47–68.

Jackson, D., 2002. Alloy: A lightweight object modelling notation. ACM Trans. Softw. Eng. Methodol. 11 (2), 256–290.

Jolliffe, I.T., 1986. Principal component analysis and factor analysis. In: Principal Component Analysis. Springer New York, New York, NY, pp. 115–128.

Kwiatkowska, M., Norman, G., Parker, D., 2011. PRISM 4.0: Verification of probabilistic real-time systems. In: Computer Aided Verification. In: LNCS, vol. 6806, Springer, pp. 585–591.

Kwiatkowska, M., Norman, G., Parker, D., Vigliotti, M., 2009. Probabilistic mobile ambients. Theoret. Comput. Sci. 410 (12–13), 1272–1303.

Martens, A., Koziolek, H., Becker, S., Reussner, R., 2010. Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In: Int. Conf. on Performance Engineering. In: WOSP/SIPEW, ACM, pp. 105–116.

Shaw, M., Garlan, D., 1996. Software Architecture - Perspectives on an Emerging Discipline. Prentice Hall.

Sobhy, D., Bahsoon, R., Minku, L., Kazman, R., 2021. Evaluation of software architectures under uncertainty: A systematic literature review. ACM Trans. Software Eng. Methodol..

Sobhy, D., Minku, L., Bahsoon, R., Chen, T., Kazman, R., 2020. Run-time evaluation of architectures: A case study of diversification in IoT. J. Syst. Soft. 159.

Warmer, J., Kleppe, A., 2003. The Object Constraint Language: Getting Your Models Ready for MDA. Addison-Wesley.

Weyns, D., Calinescu, R., 2015. Tele assistance: A self-adaptive service-based system exemplar. In: 10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS 2015, IEEE CS, pp. 88–92.