Explaining Architectural Design Tradeoff Spaces: a Machine Learning Approach

Javier Cámara[1], Mariana Silva[1], David Garlan[2], and Bradley Schmerl[2]

[1] University of York, UK
{javier.camaramoreno,mariana.silva}@york.ac.uk
[2] Carnegie Mellon University, USA
{garlan,schmerl}@cs.cmu.edu

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 distinguish the forest through the trees. In this paper, we present an approach to analyzing architectural design spaces that addresses these limitations and provides a basis to enable the explainability of design tradeoffs. Our approach combines dimensionality reduction techniques employed in machine learning pipelines with quantitative verification to enable architects to understand how design decisions contribute to the satisfaction of strict quantitative guarantees under uncertainty across the design space. Our results show feasibility of the approach in two case studies and evidence that dimensionality reduction is a viable approach to facilitate comprehension of tradeoffs in poorly-understood design spaces.

Keywords: Tradeoff analysis · Uncertainty · Dimensionality reduction.

1 Introduction

Architecting modern software-intensive systems requires exploring design spaces that are often poorly understood due to the increasing complexity and range of design choices that have to be made (and their potential interactions), as well as to the high levels of uncertainty under which these systems are expected to operate, being subject to faults, changes in resource availability and network conditions, as well as to attacks [13]. In this setting, achieving a good design that is able to guarantee the correctness of run-time system behavior while striking an acceptable balance among multiple nonfunctional properties is challenging – in particular when: (i) the context in which the system has to run contains unknown

attributes that are difficult to anticipate, and (ii) design decisions involve the selection and composition of loosely coupled, pre-existing components or services that have different attributes (e.g., performance, reliability, cost).

There are many existing approaches that help to automate the search for a good architecture and that rely on a variety of techniques such as stochastic search and Pareto analysis [1, 3, 21], as well as quantitative verification [6, 9] 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 of the relation 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 to a particular set of decisions is the satisfaction of constraints or the achievement of optimality? Which choices are correlated with others, either positively or negatively?
Providing such tool support demands investigating questions such as:

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

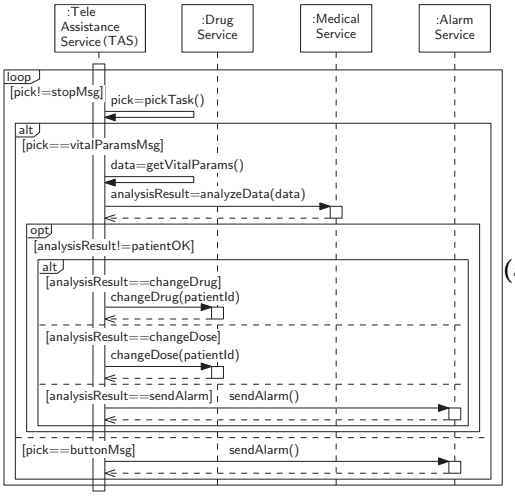
(RQ2) How much can we reduce the complexity of the information presented to the architect while preserving most of the relevant design tradeoff information?

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 a dimensionality reduction technique called principal component analysis (PCA) [16], which is typically employed to compress information in machine learning (ML) pipelines e.g., by reducing the number of features provided as input to a neural network classifier [17], as well as in natural sciences like biology to interpret high-dimensional data [20]. In our case, we combine dimensionality reduction with quantitative verification to facilitate understanding how design decisions contribute to the satisfaction of quantitative requirements across the architectural design space. 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 [9], (ii) applying PCA to tease out 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 configurations.

Our results show feasibility of the approach in two case studies and evidence that dimensionality reduction is a viable technique to facilitate understanding of tradeoffs in poorly-understood design spaces.

2 Motivating Scenario: Tele-Assistance System (TAS)

TAS [26] is a service-based system whose goal is 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 (Figure 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*.



Service name	Fail.rate (%)	Resp.time (ms.)	Cost (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.

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 (Figure 1.a). 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 (Figure 1.b).

Figure 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 [8, 9]. 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 lowest failure rates and response times.

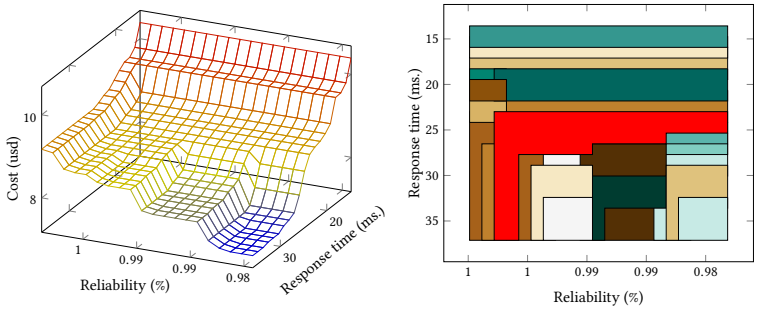


Fig. 2. TAS Analysis results.

The plot on the right is a map that shows which configurations best satisfy design criteria. Out of the set of 90 configurations that can be generated for TAS, only 24 satisfy the criteria in some subregion of the state space. If we consider that designers are interested e.g., in systems with response times $\leq$26ms and reliability $\geq$99%, we can employ the same analysis technique to determine which configurations best satisfy these constraints (highlighted in red in the figure).

Although these plots are informative and help architects to understand what specific configurations might work well in a given situation, looking at them does not facilitate understanding 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.

One of the main challenges in facilitating the understanding of the tradeoff space relates to the high dimensionality of the data and how to make it digestible 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.

3 Approach

The inputs to our approach for explaining design tradeoffs (Figure 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 output of the process is 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. The approach consists of three stages:

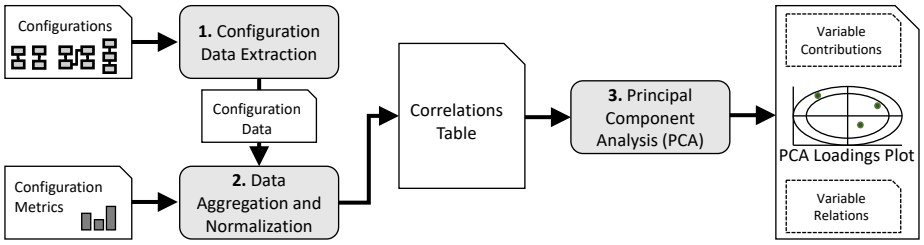


Fig. 3. Overview of the Approach.

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 architectural 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 correlations table. Table data is normalized so that all variables will have the same weight in subsequent analysis steps.
3. *Principal Component Analysis (PCA)* is employed to discover how architectural configurations differ from one another, and which variables contribute the most to that difference. Moreover, PCA enables us to discriminate whether variables contribute in the same way or a different way (i.e., are positively or negatively correlated, respectively), or if instead, are independent from each other. This enables architects to relate response variation (QoS, quantitative guarantees) to design variables.

In the remainder of this section, we first introduce some preliminaries, and follow with a detailed description of the three 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 [22] 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 $\kappa \in ArchT$. $\Pi(\kappa)$ represents the 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 [14] or OCL [25] constraints (e.g., $\forall$ w:TASWorkflowT $\bullet\exists$ a:AlarmServiceT $\bullet$ 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 i^{th} 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 an architectural 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 imposed by $\mathcal{C}_S$, 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 [9] 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* $f_{ep} : \mathcal{G}^*_{\mathcal{A}} \rightarrow \{0,1\}^{|\mathcal{A}_\Sigma|}$, returns a vector $\langle p_1, \ldots, p_{|\mathcal{A}_\Sigma|} \rangle$ 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* $f_{tc} : \mathcal{G}^*_{\mathcal{A}} \rightarrow \mathbb{N}^{|ArchT|}$, returns a vector $\langle x_{tc}(\kappa_1), \ldots, x_{tc}(\kappa_{|ArchT|}) \rangle$ encoding the number of component and connectors of each type present in a configuration. For $\kappa \in ArchT$, we define function $x_{tc} : ArchT \rightarrow \mathbb{N}$ 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 [2, 4]. Concretely, this vector is formed by concatenating the results of the following functions:
  1. *Binding presence extraction* $f_{bp} : \mathcal{G}^*_{\mathcal{A}} \rightarrow \{0,1\}^{|\mathcal{A}_\Sigma| \cdot |\mathcal{A}_\Sigma|}$ returns a vector $\langle p_{1,1}, \ldots, p_{|\mathcal{A}_\Sigma|,1}, \ldots, p_{|\mathcal{A}_\Sigma|,|\mathcal{A}|_\Sigma} \rangle$ 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* $f_{btc} : \mathcal{G}^*_{\mathcal{A}} \rightarrow \mathbb{N}^{|CompT| \cdot |CompT|}$, returns a vector $\langle x_{btc}(\kappa_1, \kappa_1), \ldots, x_{btc}(\kappa_{|CompT|,1}), \ldots, x_{btc}(\kappa_{|CompT|,|CompT|}) \rangle$ encoding the number of bindings between specific pairs of component types. For the pair of component types (κ, κ'), we define function $x_{btc} : CompT \times CompT \rightarrow \mathbb{N}$ as $x_{btc}(\kappa, \kappa') = |\{(n, n') \in \mathcal{N} \times \mathcal{N} \mid type(n) = \kappa \wedge type(n') = \kappa' \wedge bnd(n, n')\}|$.
  3. *Betweenness centrality extraction* $f_{C_B} : \mathcal{G}^*_{\mathcal{A}} \rightarrow \mathbb{R}^{|\mathcal{A}_\Sigma|}$, returns a vector $\langle C_B(\mathcal{A}^1_\Sigma), \ldots, C_B(\mathcal{A}^{|\mathcal{A}_\Sigma|}_\Sigma) \rangle$ that encodes the betweenness centrality of each node in the configuration graph, which quantifies the number of times that a node acts as a bridge along the shortest path between two other nodes [12]. Specifically, $C_B : \mathcal{A}_\Sigma \rightarrow \mathbb{R}_{\geq 0}$ is defined as: $C_B(n) = \sum_{s \neq n \neq t \in \mathcal{N}} (\sigma_{st}(n)/\sigma_{st})$, where σ_{st} is the total number of shortest paths from node s to t in the configuration graph, and $\sigma_{st}(n)$ is the number of those paths that pass through 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., configuration parameter for 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 $\mathcal{R}_\mathcal{G} = \langle r_1, \ldots, r_n \rangle, r_i \in \mathbb{R}, i \in \{1..n\}$ 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 [8], a tool that performs probabilistic model checking on collections of structural design variants that uses Alloy [15] and PRISM [19] in its backend.

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

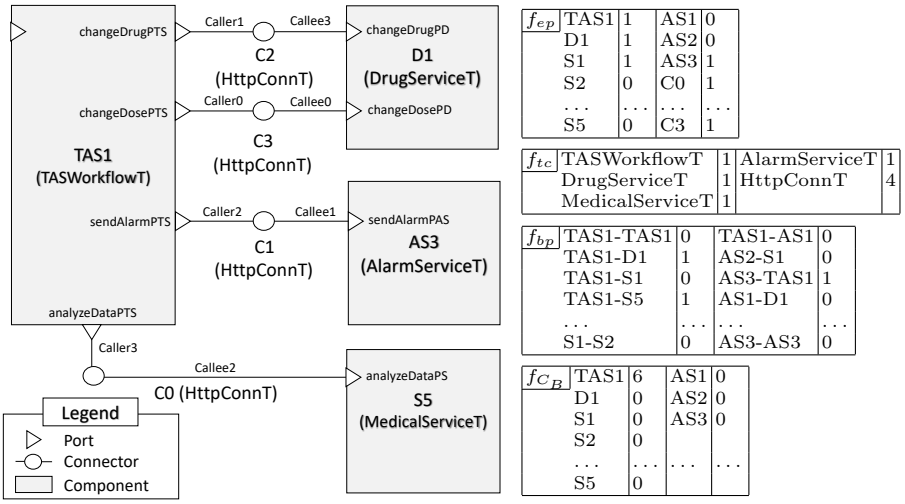
- *Data aggregation.* Given a design variable value tuple $\mathcal{D}_{\mathcal{G}}(C,T,P) \in \mathcal{DG}$, and a response variable vector $\mathcal{R}_{\mathcal{G}} = \langle r_1, \ldots, r_n \rangle \in \mathcal{RG}$ for the same configuration $\mathcal{G}$, we define the *configuration sample* 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 $x'_{i,j} = (x_{i,j} - x_j^{min})/(x_j^{max} - x_j^{min})$, where x_j^{min}, x_j^{max} are the minimum and maximum values of variable j across all samples.

Example 1. Figure 4 shows a sample TAS configuration, along with an excerpt of its encoding in the correlations table. The first and second top-most 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 the HttpConnT connector type, of which there are four instances. The two tables at the bottom describe the presence of bindings between components (f_{bp}), and their betweenness centrality (f_{C_B}). TAS is built as a service orchestration with a centralized workflow, meaning that all components but the workflow itself will have a betweenness centrality of zero.

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) [16] is a statistical projection method commonly used in ML and natural science that can help to visualize and 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 do so independently from each other. An important aspect of PCA is that it 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



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

f_{C_B}				
	TAS1	6	AS1	0
	D1	0	AS2	0
	S1	0	AS3	0
	S2	0		
	...	...	...	...
	S5	0		

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

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 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 PC contain genuine information, whereas the rest are likely to describe noise [16].

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 architectural configurations.

Example 2. The PCA loadings plot of the samples analyzed for TAS (Figure 5) displays the first two PCs, which carry a large amount of information, explaining 83% of the variance of data, with PC1 explaining the most (68%) and PC2

explaining much less variance (13%). The plot contains two ellipses that indicate how much variance is taken into account. The outer ellipse is the unit-circle and indicates 100% explained variance, whereas the inner ellipse indicates 50% of 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 to differentiate the architectural configurations.

We can observe in this case that QoS metrics like reliability, cost, and response time are all important to differentiate configurations. Out of the three, response time is the most relevant, given that it is the most important for PC1, which accounts for almost 70% of the overall variability, whereas reliability and cost are important for PC2, but comparatively have less influence overall.

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 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 Figure 2, which show that low response times and high reliability levels correspond to higher costs.

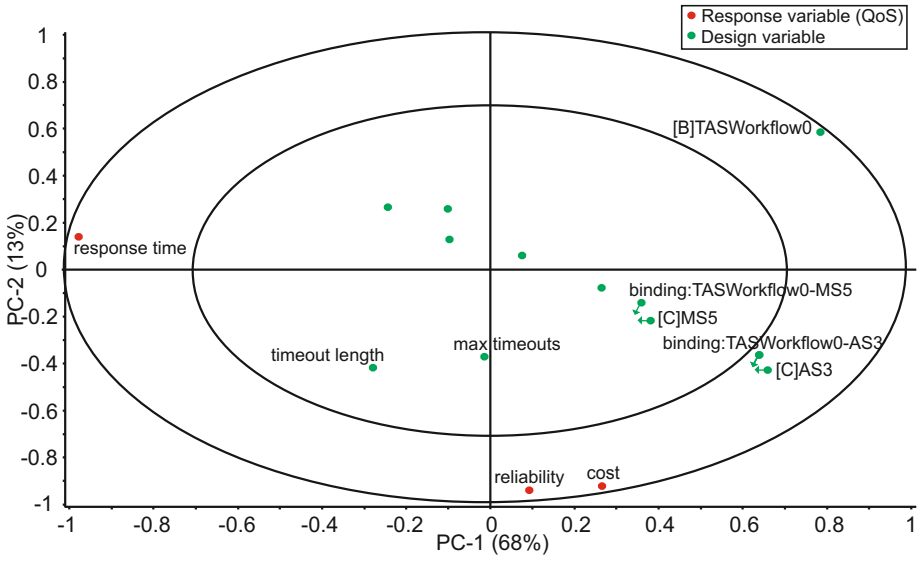


Fig. 5. Correlation loading PCA plot for TAS.

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 lower-right quadrant of the ellipse that some of the most influ-

ential 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 these 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 Figure 1.a). 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. With respect to configuration parameters, we can see that, as expected, both timeout length and maximum number of timeouts for service invocations are positively correlated with response time with respect to PC1, but also to a lesser extent with reliability and cost with respect to PC2. This observation is consistent with the fact that more service invocation retries lead to increased reliability and cost, at the expense of higher response times. Finally, the betweenness centrality of the workflow ([B]TASWorkflow0) on the right-upper corner of the diagram is a relevant variable, although it is not particularly significant in this case, given that TAS is a centralized service orchestration in which the workflow is always at the center of the composition.

4 Evaluation

The objective of our evaluation is to: (i) assess the *feasibility* of linking design decisions to requirement satisfaction (RQ1) and (ii) 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 evaluation 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 and their QoS metrics 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 [8]. Table 1 describes the number of variables and samples included in the datasets generated for the two case studies. Data analysis using PCA was performed employing "CAMO Analytics Unscrambler software (v11)" (https://www.camo.com/unscrambler/).

4.2 Scenario: Network Architecture

Architecting network-based systems that are resilient to uncontrollable environment conditions, such as network delays, or undesirable events such at virus

Table 1. Dataset dimensions for experimental evaluation.

Case Study	**# Variables**								**#samples**
	QoS	f_{ep}	f_{tc}	f_{bp}	f_{btc}	f_{C_B}	parameters	total	
Tele-Assistance System	3	10	10	27	27	10	2	89	1080
Network Architecture	2	9	4	46	11	9	3	84	2400

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 [18] 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. Ideally, the architecture of the network should: (i) minimize the probability of successful infection of high nodes in the network within 50 time units, and (ii) maximize the number of node infection or attack attempts that the virus carries out to spread itself through the high nodes.

Results The PCA loadings plot for the network architecture (Figure 6) displays the first two PCs, which explain 75% of the data variation (43% for PC1 and 32% for PC2). We can observe that both QoS metrics for the virus infection success and maximum number of virus attacks are very important to differentiate configurations. Being at the two opposite ends of the horizontal axis, they are negatively correlated, indicating that configurations that are less resilient require more virus infection attempts. Although this may sound counter-intuitive, it may be explained by the fact that the values for the virus attack success probability variable are obtained from a time-bound probabilistic analysis of the network model, meaning that scenarios in which the virus successfully infects high nodes after 50 time units are not captured in the samples. In contrast, the values for the maximum number of attacks are not time-bound.

Concerning design variables, we can observe that the most influential cluster of variables for PC1 (identified by (1) in the figure) corresponds to the bindings between low nodes and barrier nodes. Concretely, the number of bindings between barrier nodes and low nodes ([C]binding:barrierNode-lowNode) is positively correlated with the virus infection success and negatively correlated with the number of attacks. This is consistent with the fact that more bindings between low nodes and barrier nodes gives the virus more chances to spread faster to barrier nodes, compared to having to infect other lower nodes first. The second most influential cluster of variables corresponds mostly to bindings between barrier nodes (2). These variables are also positively correlated with virus infection success and negatively correlated with the maximum number of attacks. This is expected because a higher amount of bindings between barrier nodes gives more opportunities for the virus to spread, although the effect is more moderate because barrier nodes always have a higher probability of detecting virus attacks than high and low nodes. As we might expect, the number of bindings between

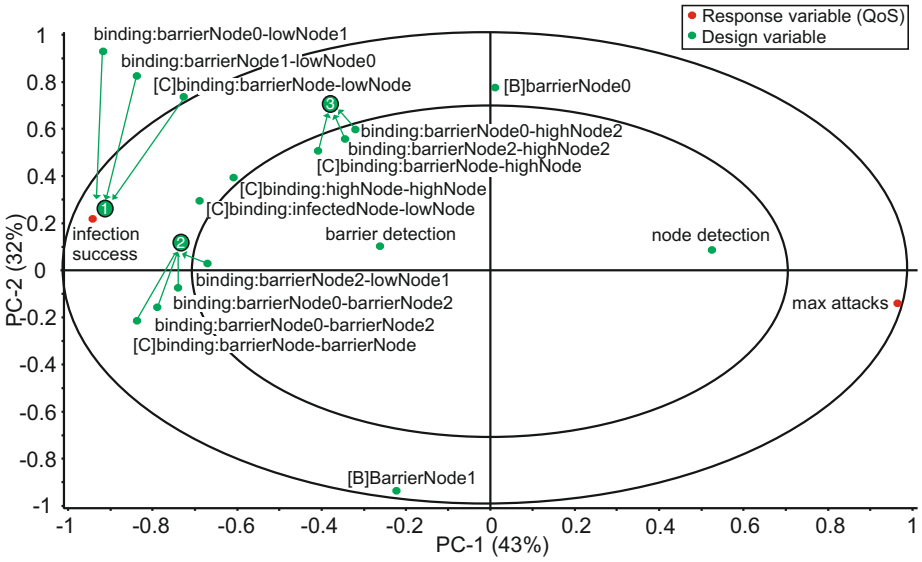


Fig. 6. Correlation loading PCA plot for the network architecture.

the node that is initially infected in the network, and the number of bindings between high nodes also influence infection success probability and maximum number of attacks in the same way, although with a more moderate impact. The cluster of variables that relates to bindings between barrier nodes and high nodes (3) is also influential in the same way, with a clear contribution both to PC1 and PC2. In PC2, we can also observe that the betweenness centrality of barrier nodes is also influential to explain variability, although there is no clear correlation with QoS variables, which are very close to PC1 and form angles close to 90 degrees with [B]barrierNode0/1. Finally, node and barrier virus detection probabilities are somewhat significant in terms of PC1, but interestingly, not much compared to other variables that are related to the topology of the configurations. This emphasizes the importance of topology for the resilience of the network, compared to attributes of individual nodes.

4.3 Discussion

(RQ1) Feasibility. PCA analysis results performed on the TAS and network architecture scenarios has shown that our approach is able to extract information that explains the relation among QoS variables and design variables. When studying the relation among QoS variables, results are consistent with observations obtained from existing analysis techniques [8, 9]. 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 studied systems. 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, it is configuration topology that explains most QoS variation. The ability of the approach to yield compelling results in both cases indicates its potential applicability to a broad range of scenarios.

(RQ2) 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 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-PC2.

We can observe that in both 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 80-93%. At the same time, the total residual variance ranges between 19 and 25%. 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.

Table 2. Information reduction and explained variance summary.

Case Study	#dataset vars.	#relevant PCA vars.	information reduction	explained variance	residual variance
Tele-Assistance System	89	6	93.25 %	81 %	19 %
Network Architecture	84	16	80.95 %	75 %	25 %

5 Related Work

Evaluation of software architectures under uncertainty is a subject that has been broadly explored [23]. 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 [1, 3, 21]. Other recent approaches to architectural design synthesis and quantitative verification [5, 8, 9] generate and analyze alternative system designs, enabling exploration of quantitative guarantees across the design space. These techniques ([8, 9] 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 [7, 10, 11, 24]. These

works employ Bayesian learning [7] to update model parameters, Model Tree Learning (MTL) to tune system adaptation logic [11], and reinforcement learning [24, 10] 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 make any claims about explicitly linking design variables with requirement satisfaction or facilitating the explainability of the design tradeoff space. Indeed, a recent comprehensive literature review on architectural evaluation under uncertainty [23] 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, 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. Our results show feasibility of the approach (RQ1) and indicate that a remarkable reduction of 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 (RQ2).

Although our approach works well in the case studies presented, PCA works optimally only in the situation where the 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, data extraction in our approach is currently limited to component-and-connector architectures with binary connectors. We will also explore extensions to the catalogue of metrics and extraction functions required to enable richer analysis of various styles of architectural representation.

Acknowledgements

This work is partly supported by award N00014172899 from the Office of Naval Research (ONR) and award H9823018D0008 from the NSA. 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 ONR or NSA.

References

1. Aleti, A., Bjornander, S., Grunske, L., Meedeniya, I.: 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 (2009)
2. Bonacich, P.: Power and centrality: A family of measures. American journal of sociology **92**(5), 1170–1182 (1987)

3. Bondarev, E., Chaudron, M.R.V., de Kock, E.A.: Exploring performance trade-offs of a JPEG decoder using the deepcompass framework. In: 6th WS on Software and Performance. pp. 153–163. WOSP, ACM (2007)
4. Borgatti, S.P.: Centrality and network flow. Social Networks **27**(1), 55–71 (2005)
5. Calinescu, R., Ceska, M., Gerasimou, S., Kwiatkowska, M., Paoletti, N.: Designing robust software systems through parametric markov chain synthesis. In: International Conference on Software Architecture, ICSA. pp. 131–140. IEEE (2017)
6. Calinescu, R., Ceska, M., Gerasimou, S., Kwiatkowska, M., Paoletti, N.: Efficient synthesis of robust models for stochastic systems. J. Syst. Softw. **143**, 140–158 (2018)
7. Calinescu, R., Johnson, K., Rafiq, Y.: Using observation ageing to improve markovian model learning in qos engineering. In: 2nd ACM/SPEC International Conference on Performance Engineering. p. 505–510. ICPE '11, ACM (2011)
8. Cámara, J.: HaiQ: Synthesis of software design spaces with structural and probabilistic guarantees. In: FormaliSE@ICSE 2020: 8th International Conference on Formal Methods in Software Engineering. pp. 22–33. ACM (2020)
9. Cámara, J., Garlan, D., Schmerl, B.: Synthesizing tradeoff spaces with quantitative guarantees for families of software systems. J. Syst. Softw. **152**, 33–49 (2019)
10. Cámara, J., Muccini, H., Vaidhyanathan, K.: Quantitative verification-aided machine learning: A tandem approach for architecting self-adaptive IoT systems. In: International Conference on Software Architecture, ICSA. pp. 11–22. IEEE (2020)
11. Esfahani, N., Elkhodary, A., Malek, S.: A learning-based framework for engineering feature-oriented self-adaptive software systems. IEEE Transactions on Software Engineering **39**(11), 1467–1493 (2013)
12. Freeman, L.C.: A set of measures of centrality based on betweenness. Sociometry pp. 35–41 (1977)
13. Garlan, D.: Software engineering in an uncertain world. In: Proc. of the Workshop on Future of Software Engineering Research, FoSER. pp. 125–128 (2010)
14. Garlan, D., Monroe, R.T., Wile, D.: Acme: an architecture description interchange language. In: Conference of the Centre for Advanced Studies on Collaborative Research. p. 7. IBM (1997)
15. Jackson, D.: Alloy: A lightweight object modelling notation. ACM Trans. Softw. Eng. Methodol. **11**(2), 256–290 (Apr 2002)
16. Jolliffe, I.T.: Principal Component Analysis and Factor Analysis, pp. 115–128. Springer New York, New York, NY (1986)
17. Khalid, S., Khalil, T., Nasreen, S.: A survey of feature selection and feature extraction techniques in machine learning. In: 2014 Science and Information Conference. pp. 372–378 (2014)
18. Kwiatkowska, M., Norman, G., Parker, D., Vigliotti, M.: Probabilistic mobile ambients. Theoretical Computer Science **410**(12–13), 1272–1303 (2009)
19. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Computer Aided Verification. LNCS, vol. 6806, pp. 585–591. Springer (2011)
20. Lever, J., Krzywinski, M., Altman, N.: Principal component analysis. Nature Methods **14**(7), 641–642 (2017)
21. Martens, A., Koziolek, H., Becker, S., Reussner, R.: Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In: Int. Conf. on Performance Engineering. pp. 105–116. WOSP/SIPEW, ACM (2010)
22. Shaw, M., Garlan, D.: Software architecture - perspectives on an emerging discipline. Prentice Hall (1996)

23. Sobhy, D., Bahsoon, R., Minku, L., Kazman, R.: Evaluation of software architectures under uncertainty: A systematic literature review. ACM Transactions on Software Engineering and Methodology (2021)
24. Sobhy, D., Minku, L., Bahsoon, R., Chen, T., Kazman, R.: Run-time evaluation of architectures: A case study of diversification in IoT. J. Syst. Soft. **159** (2020)
25. Warmer, J., Kleppe, A.: The Object Constraint Language: Getting Your Models Ready for MDA. Addison-Wesley (2003)
26. Weyns, D., Calinescu, R.: 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. pp. 88–92. IEEE CS (2015)