Distilling Architectural Design Decisions and their Relationships using Frequent Item-Sets

Stefan Sobernig
WU Vienna
Vienna, Austria
Email: stefan.sobernig@wu.ac.at

Uwe Zdun
Faculty of Computer Science, University of Vienna
Vienna, Austria
Email: uwe.zdun@univie.ac.at

***Abstract*—Much attention is paid nowadays to software architecture of a system as a set of design decisions providing the rationale for the system design. To document and share proven architectural design decisions, decisions made in concrete development projects are mined and distilled into reusable architectural decision models (a.k.a. guidance models). The available distillation approaches, however, remain ad hoc and biased towards the personal experience of few expert architects. Relationships between distilled decisions are not systematically explored. We propose an approach for distilling reusable architectural design decisions with emphasis on their relationships. Architectural knowledge artifacts (e.g., architecture documentation, interviews) are systematically coded for the occurrence of architectural design decisions and their details. Co-occurrences of coded design decisions are then processed for different relationship types using an established data-mining technique: *frequent item-sets*. The distilled relationships enter the construction of a reusable architectural decision model and contribute to organizing the design space based on empirical data (i.e., frequency patterns of co-occurrences). We report on distilling design-decision relationships from decision data collected during a three-year project on language architectures of 80 UML-based domain-specific modeling languages.**

***Index Terms*—architectural design decision; design-decision relationship; reusable architectural design-decision model; guidance model;**

I. Introduction

In recent years, software architecture is perceived as the result of a set of architectural design decisions (ADDs) intertwined with other design artifacts rather than only the system's structure using components and connectors [1], [2]. Capturing ADDs is important for analyzing, understanding, and sharing the rationale and implications of these decisions and reducing the problem of architectural knowledge vaporization [3]. Many architecture decision models and tools [4], approaches for selecting design alternatives during decision making [5], and ADD documentation approaches have been proposed in the literature: Tofan et al. identified 76 process-oriented papers as of 2014 [6]. An ambition common to all those approaches is avoiding and reducing maintenance costs by countering the vaporization of architectural knowledge [6].

Among those, approaches exist that distill common or reusable knowledge—similar to design patterns [3]—from decisions made in concrete projects to document and share proven solutions along with their forces, consequences, and (alternative) solutions (see, e.g., [7]–[12]). This distilled, reusable architectural knowledge is then offered in a *reusable architectural decision model* (RADM; also called: guidance model) including possible relationships between the decisions.

So far, the process of distilling architectural design decisions and their relationships is mostly an informal, ad hoc process based the personal experience of either only the authors of the RADM or sometimes also other designers and architects. In particular, identifying reusable decisions and decision details is driven by personal experiences, rather than systematic literature studies or harvesting readily available and documented knowledge (e.g., pattern collections) in a community [7]. This may incur unwanted biases. For example, the material entering the construction of an RADM frames any follow-up decision making based on the reused decisions, e.g. by fixating the decision makers on specific, but possibly irrelevant requirements shimmering through in this input material (framing bias; [2]). In previous work, we aimed at systematizing in particular identification of decision candidates further by performing a rigorous empirical multi-method study [11]. This study combined the results of a systematic study of the literature, interviews with industry experts, and an industrial case study.

A key issue of decision identification for reuse from architectural knowledge artifacts—whether ad hoc or systematic—is the vast amount of information which potentially enters an architect's decision inventory [7]. For example, early steps of our previous study [11] yielded more than 400 potentially relevant pattern descriptions. Even in already structured and navigable sources of reusable knowledge (e.g., a single pattern language, a reusable ADD model), there are numerous potential decision relationships to be considered. In many cases, as pattern languages and ADD models are also used outside the domain they are originally described for, potential decision relationships might significantly go beyond those documented in the pattern languages or ADD models. In our catalog on design decisions for domain-specific modeling languages [13], for example, there are 27 reusable and combinable decisions resulting in 2^{27} potential decision combinations. When populating a design-decision space from such sources, the risk is high that the effort spent on evaluating this space becomes excessive and that decision makers put emphasis on wrong or irrelevant decisions or relationships [7].

None of the existing approaches applies a fully systematic and objective approach for selecting the reusable decisions

and organizing the knowledge based on decision relationships. Frequencies of decisions and relationships in real-life projects are only informally considered by the authors of the current RADMs. Despite existing suggestions for automation support during identification [7], the degree of automation and tool support to tame the vast amount of architectural information is low, rendering approaches labor-prone, especially if larger sets of software systems need to be analysed.

In this paper, we suggest a systematic distillation approach for architectural design decisions from possibly large sets of architecting projects into an RADM, with focus on systematically harvesting relationships (associations) between reusable decisions. Key to this approach are frequent item-sets as a family of data-mining techniques that have already been explored by the software-architecture community for architecture recovery [14], [15] and for static architecture-conformance checking [16]. However, its application for distilling architectural design decisions and relations in RADMs has not been explored so far and presents particular challenges (e.g., appropriate inter-rater reliability statistics). We report on applying the distillation approach for identifying associations between design decisions on 80 architectures of UML-based domain-specific modeling languages. This application showcases the capacity of frequent item-sets to structure and to summarize an design-decision space otherwise abound in possible decision combinations (e.g., by exposing prototypical designs).

This paper is structured as follows: In Section II, we discuss background on representing ADDs and frequent item-sets. Section III introduces our distillation approach. Details of a frequent item-set analysis are then described in Section IV. The application in a large-scale distillation project is reported in Section V, followed by a discussion of limitations and lessons learned in Section VI. Related work is reviewed in Section VII, and Section VIII concludes the paper.

II. From Architectural Design Decisions to Decision-Item Sets

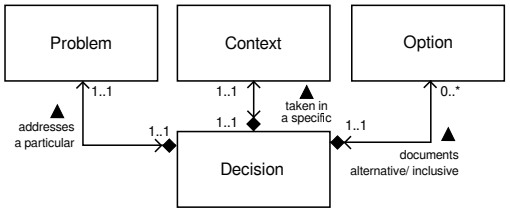

Fig. 1. Exemplary overview of architectural design decisions and their details: decision context, decision problem, and decision options.

a) ADD representation: Various approaches to reusable architecture decision models (RADM; such as [7]–[12]) use different meta-models to represent the reusable ADDs sharing the same core concepts, but currently there is no generally accepted representation [4]. In addition, any representation for documenting design rationale should be adapted to the respective domain or application. As the work presented in this paper should work together with any of those approaches, without loss of generality, we use here a simplified meta-model that only considers those core concepts and can easily be mapped to the (sometimes differently named) meta-model elements of the different RADMs.

The used core concepts and their relationships are illustrated in the meta-model in Figure 1. In particular, a *decision* is taken in a particular decision *context* (e.g., a development phase, a scope, by a certain role, or in the context of certain technology choices) to address a specific architectural design *problem* (see Figure 1). The decision and its description encompass the possible design *options* that can be adopted to solve the problem. In a decided instance of the reusable decision, one of those options would be adopted. Further details often recorded are for instance decision drivers, decision consequences, and decision states [17], [18].

b) Decision associations: An association between two decisions (or decision details) represents a possible, intentional co-occurrence of two (or more) corresponding decisions (or decision details). An association says that two or more decisions must be considered together, without implying any particular (e.g., temporal, causal) order of adoption. Decision associations can capture one decision relating to another in terms of decision drivers and decision consequences (causal sequence). More generally, a causal sequence groups decisions (decision details) which are linked pairwise by depends-on, is-excluded-by, and/or relationships [17]. When recovering a decision-making process, the time order of decision adoption can be recorded (adoption sequence), irrespective over whether time ordering subsumes any causality or not. In relevant literature, many possible decision associations are discussed, such as the influences, refinedBy, decomposesInto, forces, isIncompatibleWith, isCompatibleWith, and triggers relations in the work by Zimmermann et al. [8].

TABLE I
Sketch of important frequent item-set concepts: items, item base, transactions, and binary encoding [19], [20]

		Binary item-set encoding			
Transactions	Item sets	i_1	i_2	i_3	i_n
t_1	$\{i_2, i_3, \dots\}$	0	1	1	$\cdots$
t_2	$\{i_1, i_2, i_3, \dots\}$	1	1	1	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
t_n	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

c) Decision-item sets: In this section, we map frequent item-set concepts [19], [20] onto concepts needed for reusable decision distillation. Such distillation is based or can deliver a set of possible nominal attributions on given software architectures. Such attributions can, for instance, be guided through generic design rationale (e.g. an ADD catalog, architectural pattern collections) or the results of a (systematic) literature review (e.g. 40 software patterns relevant for the technical domain of service-based platform integration in our previous study [11]). If unguided, a base set of nominal attributions is arrived post hoc (e.g., using ex-post content coding). Each possible nominal attribute (e.g. one software pattern or decision option) is referred to as a (decision) *item*. All available

nominal attributes can be encoded as a set of binary attributes $B = \{i_1, i_2, \ldots, i_n\}$ called the *item base* (see also Table I).

A coding procedure on architectural knowledge artifacts—for details see Section III—yields a collection of item sets referred to as the data base T, with each recorded item set being a *transaction*; $T = \{t_1, t_2, \ldots, t_n\}$. The *cover* K of any item set $I \subseteq B$ indicates the transactions in B it is contained in: $K = \{t | t \in T \wedge I \subseteq t\}$. The support s (a.k.a. absolute frequency) of a given item set is the cardinality of its cover: $s = |K|$; that is, the number of containing transactions. Such decision-item sets are eligible to represent *decision associations* (see above).

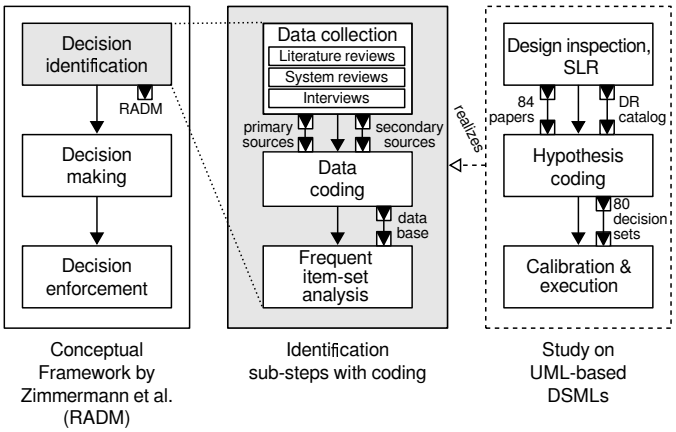

Fig. 2. An overview of the proposed decision-identification and distillation approach as part of a conceptual decision-modeling framework based on reusable architecture decision models (RADM, GM; [7]). Its application to the domain of UML-based DSMLs [13] is elaborated on in Section V.

III. Approach Overview

To put our approach into a broader context, we consider it as a refinement of the decision process based on reusable architecture decision models (RADM) by Zimmermann et al. [7]. This approach claims that a decision process usually has three major steps: decision identification, decision making, and decision enforcement. Figure 2 depicts these RADM-based steps in the left-most column, followed by our suggested identification sub-steps, and an instantiation of these sub-steps in an actual distillation project (described in Sections IV and V). Zimmermann et al. [7] replace the decision identification step, specific to each project, with an identification resulting in a set of reusable decisions. In their *decision identification* the objective is to identify required decisions based on knowledge artifacts relevant for the technical domain at hand and likely to document recurring decision contexts, problems, and solution candidates. The outcome of this step is the RADM for the actual *decision making* in the next step. Making a decision based on the identified candidates involves an evaluation of decision drivers such as domain- and project-specific requirements or quality attributes. Finally, *decision enforcement* is about communicating decisions made to stakeholders and executing them. The goal of reusing decisions from an RADM is to increase productivity significantly and have a positive effect on quality [7]. The focus of our work is to systematize the reusable decision-identification step in Zimmermann et al.'s decision process. Note that our approach can easily be combined with other RADM approaches (e.g., [8]–[12]).

Our approach has three major steps: collection, coding, and frequent item-set analysis (see middle column in Figure 2). Data collection refers to the architects in a software development project gathering as many architectural decisions and solutions in an architectural solution domain as possible with (direct) or without involvement (indirect) of peers [21]. Direct data-collection techniques include interviews, questionnaire surveys, brainstorming, and focus groups. Indirect techniques are primarily architecture-documentation analyses including literature reviews (e.g., systematic literature reviews as in [11], [13]), analyses of process documentation (project logs, work diaries), if available, and reviews of system artifacts (e.g., code and test bases).

Data collection may yield two types of architecture knowledge artifacts (see Figure 2): A *primary source* is a descriptive piece of documentation on a particular architecture authored by the original architects themselves. In [13], for example, we collected 84 scientific publications using a mixed engine- and citation-based search as primary studies on existing architectures. A *secondary source* is a reflective and/or prescriptive piece of documentation on architecting for a particular domain or across domains. A scientific literature review, e.g., can detect secondary sources in terms of literature surveys, systematic reviews, pattern collections, opinion papers, personal experience reports, and evaluation research [22]. In [11], e.g., we performed a manual and subjective systematic literature review for pattern collections. The availability of secondary sources determines the options for data coding (open vs. guided; see below).

During data coding, the primary sources collected so far are subjected to a systematic, qualitative content-coding procedure [23], [24]. Coding involves one or more experts searching for occurrences of architectural design decisions or solutions in a set of primary sources and to document them. The main objective of coding is to reduce the enormous amount of predominantly qualitative data obtained from data collection. A key coding step is developing a *coding schema*. A coding schema, which is defined before actual coding based on existing concepts, realizes *guided* (or deductive) coding. This is based on previous, extrinsic knowledge, such as existing theories (e.g., reference architectures, prescriptive guidelines), reports on prior research in the domain, or data-collection artifacts (e.g., interview guides). A guided coding approach assumes secondary sources being available from the data collection step. When a coding scheme is built by creating categories and sub-categories based on the primary sources in an incremental manner during coding, as secondary sources are not available at all, this is referred to as a procedure of *open* (or inductive) coding. Available open-coding techniques are summarizing, subsumption, and open/selective coding as in grounded-theory approaches (see [23] for an overview). Whether guided or open (or a combination), the output of

TABLE II

A CONVENIENCE SAMPLE OF TEN DSMLS EXTRACTED FROM THE STUDY DATA FROM [13] FOR ILLUSTRATION PURPOSES. EACH DSML ARCHITECTURE WAS CODED FOR THE PRESENCE OF ADDS GROUPED INTO SIX DIFFERENT DECISION GROUPS (EXPANDED CODE, ABBREVIATED CODE): LANGUAGE-MODEL DEFINITION ($d1$), LANGUAGE-MODEL FORMALIZATION ($d2$), LANGUAGE-MODEL CONSTRAINTS ($d3$), CONCRETE-SYNTAX SPECIFICATION ($d4$), BEHAVIOR SPECIFICATION ($d5$), PLATFORM INTEGRATION ($d6$).

DSML	Transaction	
	expanded	abbreviated
CompSize	{LANGUAGE-MODEL DEFINITION, LANGUAGE-MODEL FORMALIZATION, CONCRETE-SYNTAX SPECIFICATION}	$\{d1, d2, d4\}$
EIS	{LANGUAGE-MODEL DEFINITION, LANGUAGE-MODEL FORMALIZATION, LANGUAGE-MODEL CONSTRAINTS, CONCRETE-SYNTAX SPECIFICATION}	$\{d1, d2, d3, d4\}$
UACL	{LANGUAGE-MODEL DEFINITION, LANGUAGE-MODEL FORMALIZATION, LANGUAGE-MODEL CONSTRAINTS, CONCRETE-SYNTAX SPECIFICATION}	$\{d1, d2, d3, d4\}$
MoDePeMART	{LANGUAGE-MODEL DEFINITION, LANGUAGE-MODEL FORMALIZATION, CONCRETE-SYNTAX SPECIFICATION}	$\{d1, d2, d4\}$
UML-GUI	{LANGUAGE-MODEL DEFINITION, LANGUAGE-MODEL FORMALIZATION, PLATFORM INTEGRATION}	$\{d1, d2, d6\}$
SMF	{LANGUAGE-MODEL DEFINITION, LANGUAGE-MODEL FORMALIZATION, CONCRETE-SYNTAX SPECIFICATION}	$\{d1, d2, d4\}$
BIT	{LANGUAGE-MODEL DEFINITION, LANGUAGE-MODEL FORMALIZATION, LANGUAGE-MODEL CONSTRAINTS, CONCRETE-SYNTAX SPECIFICATION, PLATFORM INTEGRATION}	$\{d1, d2, d3, d4, d6\}$
UML-PMS	{LANGUAGE-MODEL DEFINITION, LANGUAGE-MODEL FORMALIZATION, LANGUAGE-MODEL CONSTRAINTS, CONCRETE-SYNTAX SPECIFICATION}	$\{d1, d2, d3, d4\}$
SECRDW	{LANGUAGE-MODEL DEFINITION, LANGUAGE-MODEL FORMALIZATION}	$\{d1, d2\}$
UML4SOA	{LANGUAGE-MODEL DEFINITION, LANGUAGE-MODEL FORMALIZATION, LANGUAGE-MODEL CONSTRAINTS, CONCRETE-SYNTAX SPECIFICATION, PLATFORM INTEGRATION}	$\{d1, d2, d3, d4, d6\}$

this step is a data base of decision-item sets (transactions; see also Section II).

Finally, via running a *frequent-item set analysis* [19], the coded decision data are processed to highlight a subset of the identified decisions and their details based on their relevance to the decision-making context. The objective is to prepare not a complete, but a tailored design-decision space for the decision-making step. The main outcome of the frequent-item set analysis step is an RADM refined by data on occurrences and co-occurrences of its model elements.

Toolkit: The analysis step is supported by a tool using existing statistical software components in R, mainly the R package `arules` [20]. The toolkit provides reusable frequent item-set definitions (e.g., generators) and a battery of special-purpose inter-rater agreement statistics (e.g., Kuppner-Hafner index, Krippendorff's extended alpha).

IV. ADD DISTILLATION USING FREQUENT ITEM-SETS

In this section, ADD distillation based on frequent-item set analysis is shown by looking at a concrete data example from an actual distillation project [13]. Along the way, we introduce key concepts of frequent-item set analysis (e.g., support, closedness, maximality, freeness) based on the background in Section II. In [13], we realized the model procedure depicted in Figure 2 (right-most column): Documentation on 80 DSML architectures was collected via a combination of design inspections and a large-scale systematic literature review (SLR). The documentation content was coded using a procedure of guided hypothesis coding yielding a data base of 80 decision sets. The subsequent frequent item-set analysis was calibrated—among other things—by defining two different abstraction levels for ADDs based on the research questions: decision groups representing critical decision points (e.g., LANGUAGE-MODEL DEFINITION, CONCRETE-SYNTAX SPECIFICATION; [25]) and actual decisions within these groups (FRONTEND-SYNTAX EXTENSION, MIXED SYNTAX etc. for decision point CONCRETE-SYNTAX SPECIFICATION). See Section V for the study details. As an illustrative example, 10 out of 80 decision sets (transactions) at the level of decision groups are reproduced in Table II. In the remainder, we refer to the decision groups and these 10 transactions in an abbreviated notation for the sake of readability (e.g., $d1$, $\{d1, d2\}$; see Table II for the mapping).

The task of identifying frequent patterns of item sets is specific to a given *item base* of decision codes (see Section II). In the example, the item base consists of the six codes representing decision groups relevant in designing a DSML: $d1$–$d6$. Another input to the analysis is the collection (data base) of 10 *distilled* sets of decision codes (transactions) forming a data base. Each transaction represents a complete DSML design. Consider the example of UML4SOA [27], one of the 80 third-party UML-based DSMLs reviewed. UML4SOA is a DSML for modeling service-oriented architectures based on tailored SoaML and UML activity, class, and component diagrams. Its language architecture was found to result from ADDs falling into five decision groups: LANGUAGE-MODEL DEFINITION, LANGUAGE-MODEL FORMALIZATION, LANGUAGE-MODEL CONSTRAINTS, CONCRETE-SYNTAX SPECIFICATION, and PLATFORM INTEGRATION; or $\{d1, d2, d3, d4, d6\}$ in short (see Table II). Any transaction is a subset of the item base. In the example, hence, there are 16 possible, unique item sets which can be expressed using the six decision groups. The resulting, potential design-decision space of 16 item sets is visualized as a Hasse diagram [26] in Figure 3.

By studying the data base of ten transactions alone, we arrive at three initial and immediately useful observations:

Uniquely distilled item sets: In the data base, there are five *uniquely* distilled item sets. See the corresponding five nodes in the Hasse graph in Figure 3, represented by solid rectangles. Conversely, there are eleven out of 16 potential item sets which cannot be found in the collection as-is (see the dashed rectangles in Figure 3).

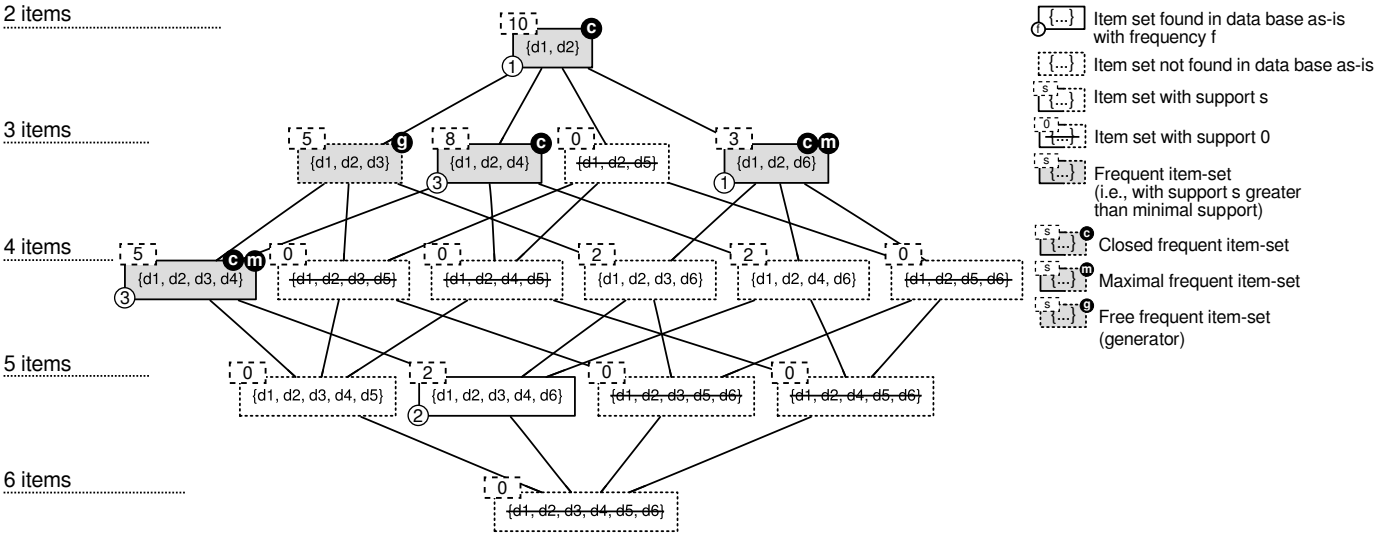

Fig. 3. Visualization of the design-decision space of decision points spawned by the item base as a Hasse diagram [26]. A Hasse diagram visualizes a hypothesized relationship structure between item sets in terms of a partially ordered set: $\{O : O \in (\{d1, d2, d3, d4, d5, d6\}, \subseteq) \wedge \{d2\} \subset O \wedge \{d1\} \subset O\}$. Edges point downwards, omitting edge directions. Information from a frequent-item-set analysis using the base of ten observed item sets in Table II is superimposed onto the Hasse graph.

Repeatedly distilled item sets: Each of the five unique item sets has at least one or more occurrences in the database. For example, there are three DSMLs sharing the item set $\{d1, d2, d4\}$ (i.e., CompSize, MoDePeMART, and SMF). On the other hand, we find only one DSML (SECRDW) with decisions for two decision groups: $\{d1, d2\}$.

Cardinalities: By looking at the cardinalities of uniquely distilled item sets, we learn about, e.g., the minimum number (two) or the maximum number (five) of decision groups present in the collection of ten DSMLs. No DSML architecture resulted from decisions of all six decision groups.

We can gain additional insights from contrasting the distilled item sets to the hierarchical structure of possible item subsets. On the one hand, by considering the distilled sets alone, we do not learn about characteristic subsets of items being recurring sets throughout the collection of the ten exemplary DSMLs. For example, while the item set $\{d1, d2\}$ has been found to represent a concrete DSML design (SECRDW; see above), we omit how often this set re-appears as a proper subset of the remaining nine item sets. On the other hand, we do not learn whether there are other patterns of items re-occurring as characteristic subsets only throughout the distilled data base on DSMLs. In Figure 3, this is exemplified by the item set $\{d1, d2, d3\}$, which does not characterize a single DSML design as-is (i.e., it is not contained by the data base), but is shared as item-set fragment by five DSMLs.

The *support* s of a given item set is expressed as the number of transactions in the data bases in which it is contained as a subset (see Section II). The support can be computed for all 16 possible item sets on the design-decision space depicted in Figure 3. For example, the support of item set $\{d1, d2\}$ amounts to 10 (in absolute terms; relative support is 10/10 or 1). In fact, this subset is contained by all ten DSMLs while there is only one DSML which is described by this item set exactly. A support of 0 indicates that an item set is not present as-is in the collection *and* that it is not contained by any item superset residing at the next-lower levels of the design-decision space (assuming the top-down ordering in Figure 3). Consider the example of $\{d1, d2, d5\}$. There is no DSML which is exactly characterized by these three decision groups and there is no DSML which contains this subset. This notion of support allows for applying a number of restrictions on the item sets: minimum support (frequency), closedness, maximality, and freeness. By combining these restrictions when filtering the total number of item sets expressible (a.k.a. design-decision space), we can identify four types of item sets: frequent item-sets (in the strictest sense), frequent-closed item sets, maximal-frequent item sets, and free-frequent item sets.

Frequent item sets [19], [28]: Typically, we are interested in finding item sets out of the total design-decision space which have a *minimum support*. Minimum support reflects the requirement that a given item set must occur or be contained by a minimum number of item sets, i.e. in our case, DSMLs. All (distilled and possible) item sets having a support s equal to or greater than the minimum support s_{min} are called *frequent* item sets. In our running example based on the DSMLs in Table II, we apply a $s_{min} = 3$. An option is frequent if it has a support of 3 or more, that is, it is found in at least three different DSML projects. This results in five frequent item sets in our example (see also the grey rectangles in Figure 3): $\{d1, d2\}$, $\{d1, d2, d3\}$, $\{d1, d2, d4\}$, $\{d1, d2, d6\}$, and $\{d1, d2, d3, d4\}$.

Frequent-closed item sets [19]: An item set is said to be *closed* if it is frequent and if none of its proper item supersets has a support equal to or less than the support of this item set. In our example, we find four item sets out of

the five frequent ones in this condition: $\{d1, d2\}$, $\{d1, d2, d4\}$, $\{d1, d2, d6\}$, and $\{d1, d2, d3, d4\}$ (see the nodes marked "c" in Figure 3). The item set $\{d1, d2, d3\}$ is not closed because its $\{d1, d2, d3, d4\}$ has the same support of 5. That is, an item set is closed if no proper item superset containing a given item set is contained by the distilled item sets, in which the item set is contained, in the data base. This can be the case under three conditions (i–iii) important to recovering decision associations:

(i) A frequent item set corresponds to at least one transaction or design as-is. An example is $\{d1, d2\}$ for SECRDW. In this case, none of its supersets (e.g., $\{d1, d2, d4\}$) can naturally be part of this item set. Conversely, $\{d1, d2, d3\}$ is not closed because it does not appear as-is in the collection, only as a subset of $\{d1, d2, d3, d4\}$ (e.g., for EIS and UACL) and of $\{d1, d2, d3, d4, d6\}$ (i.e., for BIT and UML4SOA).

(ii) A frequent item set represents the *least-common* item subset for (some of) its proper item supersets as contained in the distilled item sets of the collection: Consider extending the example based on Table II and Figure 3. In its given setting, $\{d1, d2, d3\}$ is not closed (see above). If *one* DSML is added which is described by the distilled item set $\{d1, d2, d3, d5, d6\}$, then $\{d1, d2, d3\}$ would become closed because it qualifies as the least-common subset contained in both the five transactions containing $\{d1, d2, d3, d4\}$ and the newly added one. Formally, this would be reflected in an increased support of $\{d1, d2, d3\}$ (6), therefore, surpassing the support of $\{d1, d2, d3, d4\}$ (5).

(iii) Both conditions 1) and 2) above hold for a frequent item set: This is the case for $\{d1, d2\}$. First, it appears as-is as an distilled item set (SECRDW; see Table II). Second, it turns out to be the least-common frequent subset for the transactions containing d3/d4 *and* d6 (i.e., BIT and UML4SOA).

In summary: All closed item sets are frequent ones. The subset of closed item sets can be smaller than the number of total frequent subsets (i.e., four of five item sets in our example). Non-closed frequent item sets are subsets of one or several closed item supersets.

Maximal-frequent item sets [19]: The set of frequent item sets is a subset of the design-decision space which represents minimum support (or adoption of certain options) in the studied DSML projects. This subset, however, contains redundant information. For example, $\{d1, d2\}$ is included by all other four frequent item sets which are proper supersets of the former. Any of these supersets represent the condition of $\{d1, d2\}$ being frequent. A frequent item set is called *maximal* if none of its proper subsets is frequent (i.e., has equal to or more than the minimum support). This notion is suitable for removing the redundancy by upward containment between frequent item sets and to establish a potentially smaller subset of characteristic frequent item sets which is capable of representing all other frequent item sets.

In our example, we find two maximal-frequent item sets: $\{d1, d2, d6\}$ and $\{d1, d2, d3, d4\}$ (see also the nodes marked "m" in Figure 3). The remainder of three frequent item sets are all subsets of these two item sets. The maximal subset of the set of frequent item sets, therefore, exhibits those frequent item sets with maximum cardinality (three and four decision groups, respectively). From a design-decision-space perspective, a maximal item set reflects a frequent combination of a maximal number of decision groups considered jointly—besides summarizing the entire sub-space of frequent item sets. Applied to the ten DSMLs, we can therefore state that a critical number of DSMLs result from ADDs in three and four decision groups, but never in five or six. In addition, based on this sample, we could summarize that most frequently DSMLs either take decisions at decision groups $d3$ and/or $d4$ or—mutually exclusive—at $d6$, if any decision beyond $d1$ and $d2$ were taken at all. All maximal item sets are closed. Therefore, the set of maximal item sets is a subset of the closed subset of item sets.

Free frequent item sets [19], [28]: An item set is considered a free item set (a.k.a. generator) if it is the minimal subset (i.e., the smallest in terms of items contained) among all the item subsets appearing in a transaction. It is minimal in the sense that there are no smaller item sets (i.e., the proper subsets of the free set) which appear as-is in a transaction. A free item set (generator) is frequent when having at least minimum support.

Of particular interest to us are the free item sets which form the closed frequent item sets as found for the selected DSMLs. As stated above, the closed frequent item sets serve as a compact representation of the entire distilled frequent design space (i.e., all frequent item sets can be expressed as subsets of the closed item sets). However, as the largest frequent building blocks (in terms of items contained) found in reviewed DSMLs, they are not as selective when characterizing transactions. For instance, to find the transactions containing $d3$, we take the closed set $\{d1, d2, d3, d4\}$ marked in Figure 3, and match it against the ten distilled item sets. This will yield five item sets. This result, however, contains noise because the five item sets are also those containing $d3$ jointly with $d4$.

According to the above definition, $\{d1, d2, d3\}$ is found to be a frequent generator of the closed set $\{d1, d2, d3, d4\}$ in the sense that it is capable of matching all distilled item sets while being of smaller size in terms of decision items. By being smaller, it is more informative because it can be more easily combined, for example, with other smaller closed or generator sets (e.g., $\{d1, d2, d4\}$ in Figure 3) to describe the design-decision space. Combining the comparatively larger closed sets as descriptors suffers from more redundancy, such as $\{d1, d2, d4\}$ and $\{d1, d2, d3, d4\}$ differing only by one option. It also follows from the above definition that a free item set or generator cannot correspond to an entire transaction (design); it is a building block only.

V. Reusable Architecture Rationale for DSMLs

In the following, we highlight key steps and the most important findings of applying our distillation approach on decision data obtained from a 3 year research project on language architectures for UML-based domain-specific modeling languages (DSML) [13]. In this context, a DSML architecture is considered the fundamental structure formed by key artifacts

TABLE III
SCOPING AND CALIBRATION OF THE CONCEPTS OF A FREQUENT ITEM-SET ANALYSIS FOR THE DISTILLATION OF REUSABLE DESIGN DECISIONS FOR UML-BASED DSMLS [13].

Generic	Applied (DSML)
(Decision) Item	Items represent either a) 27 individual decisions from [13] or b) six decision groups capturing points of decision making [25].
Transaction	Set of category codes for a) or b) above per DSML
Data base	The collection of sets of decision items recorded for 80 DSMLs
Support (abs., rel.)	Number (percentage) of DSML designs in which a given option (or subset of options) was adopted
Minimum support (abs., rel.)	Decision option and option sets having at least reported applications in *three* (3) different third-party DSMLs, that is, DSMLs not developed by the authors; motivated by the least three known uses of a pattern in existing software systems (see, e.g., [29])

and ADDs yielding them such as its abstract syntax, abstract-syntax constraints, concrete syntax, its behavior specification, transformations for platform integration, and modeling tool support [25]. This research project comprised the distillation steps as outlined in Figure 2: data collection, data coding, and a calibrated frequent item-set analysis (see also Table III).

Data collection was sequential: In preparing the project, we inspected 10 DSML architectures from our own research group to gather decision candidates (supported by secondary studies on DSML development and UML extensions). To collect additional and unbiased evidence (primary and secondary sources), we then designed a large-scale SLR. First, we compiled a corpus of 37 reference publications ("quasi-gold standard") from key venues on DSMLs and UML extension techniques. Second, this corpus guided an automated search for primary DSML-specific publications by deriving search terms from its metadata and fulltext content. The search was executed in four search engines: SpringerLink, IEEE Xplore, Scopus, and ACM Digital Library. Third, we ran a manual snowballing search based on the finally selected publications from the main and automated search. We ended up considering more than 8,000 search hits for the years between 2005 and 2012, from which 84 conference and journal publications documenting 80 unique DSML architectures were finally selected to enter the coding step. 10 of those DSMLs have been used for the running example in Section IV; for a list, please refer to [13].

Data coding using a variant of deductive qualitative content coding (*hypothesis coding* [24]) on the primary sources collected via the above SLR: 84 scientific papers and their auxiliary design-documentation artifacts, e.g. package diagrams as well as implementation artifacts such as metamodel, profile, and concrete-syntax specifications. Coding was performed by the 3 study authors as coders (pairwise, blinded for the alter's coding). The coders were guided by a coding scheme developed before the fact, which provided indicators and decision rules for coders. Coding was performed in the typical steps: The total material was segmented into themes (according to the six decision points already introduced in Section IV) and coding units (complete phrases, phrase blocks, and content items such as tables, figures, listings, and formula blocks). During main coding, each unit of coding was assigned to one of 27 categories (decision options) of the coding scheme; first as text marks in the respective and segmented document. The 27 available categories corresponded to 27 reusable ADDs documented in a draft ADD catalog, as another outcome of the above SLR. Coding yielded 80 sets of such assigned categories, one for each DSML architecture.

To illustrate the outcome of this coding step, consider the example of UML4SOA [27]. The language model of UML4SOA is defined textually (INFORMAL TEXTUAL DESCRIPTION; coding category: 1.1) and integrates with the UML via a UML metamodel extension (METAMODEL EXTENSION; 2.3) as well as equivalent UML profile definitions for tool adoption (PROFILE RE-/DEFINITION; 2.2). In addition, the metamodel extension and profile definitions are accompanied by OCL constraint definitions (CONSTRAINT-LANGUAGE EXPRESSION; 3.1). The metamodel extension comes with new and resampled diagram symbols (DIAGRAMMATIC SYNTAX EXTENSION; 4.2), the profiles imply model annotations (e.g., comments containing tags; MODEL ANNOTATION; 4.1) and symbol reuse (DIAGRAM SYMBOL REUSE; 4.6). As for platform integration, UML4SOA employs an INTERMEDIATE MODEL REPRESENTATION (6.1) to transform extended UML activities in several steps (M2M TRANSFORMATION; 6.5) into web-service orchestration specifications (BPEL) using API-based generators (e.g., the Eclipse/EMF Java API; API-BASED GENERATOR; 6.3). The resulting decision-item set for UML4SOA is therefore: $\{1.1, 2.2, 2.3, 3.1, 4.1, 4.2, 4.6, 6.1, 6.3, 6.5\}$.

Frequent item-set analysis: The analysis was calibrated as summarized in Table III. We learned that the reviewed DSMLs have a maximum of ten decisions per DSML architecture. For recurring item subsets, i.e. item subsets found in more than one DSML, the maximum number of items in a particular item subset was seven. At the level of individual decision points, these maxima translate into frequently recurring subsets containing options from three decision records only: language-model definition (d1), language-model formalization (d2), and concrete-syntax definition (d4). Beyond such basics, we obtained insights on *prototypical designs* and *characteristic combinations* of decisions.

Prototypical designs: In this study, a prototypical design was defined as frequent item set which represents a largest item subset (here: DSML architecture fragment) which was also frequently found to represent a complete transaction (here: DSML architecture; see also Section IV). This prototype item-set is frequently found extended by adding other (frequently or infrequently observed) options. In this sense, it represents an *evolutionary* prototypical design to derive extended DSML architectures. The notion of prototype item-sets matches DSML

design practices commonly described and reflected on in secondary literature on extending the UML and UML-based DSL development (see Table IV): UML extension, UML specialization, and UML piggybacking using UML profiles.

This notion of prototype item-set is particularly useful for structuring the design space described by the 24 observed decision options. First, they cover a critical share of the 80 DSMLs. Second, they stress commonalities and differences in terms of decision options between these highly representative option combinations. In particular, the seven prototype item-sets characterize 30% of the studied DSMLs architectures (24/80) in their entirety. Furthermore, they are contained as large proper subsets by 25 extended option sets; therefore reaching a total coverage of approximately 61% of the DSML architectures included in our study (49/80). By looking at the common and varying decision options in the seven prototype item-sets, we find that all seven are combinations of nine decision options. These nine decision options correspond to the leaf elements of the feature diagram in Figure 4.

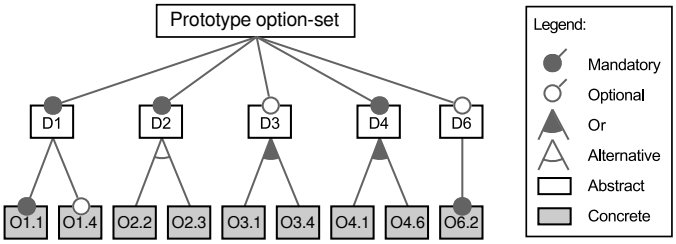

Fig. 4. A feature diagram representing the seven prototype item-sets found in the pool of 80 third-party DSML architectures . These seven prototypical designs include nine different decisions. Each of the seven distilled prototype item-sets listed in Table IV is one of the possible, valid configuration of this feature space.

Combinations: Characteristic combinations of design-decision options were encoded as frequent item set which is also a smallest (i.e. of minimal size) recurring *proper* item subset contained by observed DSML designs and/or by observed design fragments. We distinguished between two kinds of smallest common item subset: (1) two item subsets specific to one decision record (d1–d6); (2) seven item subsets specific to two or more decision records (d1–d6).

For example, as for platform integration (d6), we found a subset $\{\text{GENERATOR TEMPLATE}, \text{M2M TRANSFORMATION}\}$ ($\{6.2, 6.5\}$, support: 3) that indicates that the respective 3 DSMLs use a two-level model transformation chain (PIM-PIM-PSM): First, platform-independent models (PIM) are transformed into another PIM representation which is then transformed into a structured textual, platform-specific (PSM) representation. One of them, UML2Alloy, extends UML class models (PIM) which are transformed into models of an Alloy metamodel (PIM), which are finally transformed into textual Alloy definitions accepted by an Alloy model checker (PSM).

VI. Discussion

Limitations: Our approach inherits the limitations characteristic for predominantly qualitative field studies (e.g., documentation, literature, and code reviews) used for data collection and content analyses. We elaborate on threats and mitigation strategies in [11], [13].

Any content analysis on the primary and secondary sources is situational and requires interpretation to identify ADD details (e.g., category assignments), and different interpretations (depending on the personal bias of the content analysts) may be equally valid. Therefore, errors in conducting a content analysis (e.g., applying a coding scheme) risk being perceived as an alternative valid interpretation. As a counter measure, the interpretations must be compared between persons and/or across time (e.g., double-coded, re-coded). The degree of (in-)consistency between persons/ across time must be reported to the reader (inter-rater reliability, IRR). Coding data on ADDs turned out to have specific requirements on inter-rater reliability measurement. For example, the structure of coding schemes (main categories, sub-categories) yields multi-attribute data sets, which are not natively supported by standard IRR statistics (e.g., Cohen's Kappa). We, therefore, identified and implemented adequate IRR statistics (Kupper-Hafner Index, extended Krippendorff alpha) in our toolkit.

An important threat specific to content coding is that the coding schemes (whether used deductively or inductively) do not capture what they are set out to capture (e.g., it leads to missing entire ADDs or decision details). For guided (deductive) coding in [13], we assessed the content validity using expert evaluation, having non-involved experts on DSMLs review the coding scheme. For open (inductive) coding as in [11], we monitored the assignment frequencies across categories and the assignments to residual categories. These can be signs of incomplete or undifferentiated coding schemes [23].

There is a critical trade-off between a quantitative analysis of co-occurrences (frequent item-sets) and characterizing distilled ADDs and their relations. On the one hand, a systematic decision identification is meant to yield a substantial amount of decision material to avoid, e.g., anchor and framing biases [2]. On the other hand, when reducing this content base to a manageable size via coding, one consciously neglects data on certain decisions and abstracts from the actual decisions.

Equally important, the nature of any ADD relationship (e.g., causal, temporal) is lost. This is not only due to coding and the representation choice of decision-item sets. Whether information on the order of ADDs, for example, can be distilled depends on the data-collection technique [21]. By applying a direct inquisitive or observational technique (e.g., interviews, participant observation), the ordering information can be gathered. By applying an indirect technique such as a kind of documentation analysis, as in our study, information on ordering often remains implicit and, therefore, unrecoverable. Similar, even if documented, the indirectly observed order of options might also follow from biases such as presentation requirements, e.g., of an scientific publication. Although such abstraction is a limiting factor, the benefits of frequent item-sets are elsewhere: They help render the design space for decision makers more accessible (e.g., by highlighting frequently adopted decisions) and present cues on otherwise hidden

TABLE IV
OVERVIEW OF THE SEVEN PROTOTYPE ITEM-SETS (ORDERED BY DECREASING ABSOLUTE SUPPORT). DETAILS ON THE EXEMPLARY DSMLS INCLUDING CITATIONS ARE DOCUMENTED IN [13].

Prototype	Item set	Support (abs.)	Frequency (abs.)	DSMLs (ex.)
UML piggybacking plus informal constraints	$\{1.1, 2.2, 3.4, 4.1, 4.6\}$	30	5	UML-AOF, PredefinedConstraints, UML-PMS
UML piggybacking plus formal constraints	$\{1.1, 2.2, 3.1, 4.1, 4.6\}$	26	4	REMP, CUP, UML4PF
Two-level UML piggybacking	$\{1.1, 1.4, 2.2, 4.1, 4.6\}$	22	5	SPArch, MoDePeMART, RichService
UML piggybacking for domain-specific M2T system	$\{1.1, 2.2, 4.1, 4.6, 6.2\}$	15	3	DPL, WCAAUML, WS-CM
UML piggybacking plus mixed constraints	$\{1.1, 2.2, 3.1, 3.4, 4.1, 4.6\}$	13	3	ArchitecturalPrimitives, SHP, C2style
UML metamodel ("middleweight") extension	$\{1.1, 2.3, 4.6\}$	10	4	UML2Ext, UML4SPM, MDATC
Two-level UML piggybacking plus mixed constraints	$\{1.1, 1.4, 2.2, 3.1, 3.4, 4.1, 4.6\}$	5	3	UACL, SafeUML, and IEC61508

relationships between ADDs to be characterized qualitatively.

Lessons Learned: Decisions and decision relationships distilled using frequent item-sets are immediately useful devices to structure and analyze a design space during decision identification (see Section III). In addition, we learned that the distilled frequency data can be used to render an RADM *tailorable* [30]. For example, the study in [13] resulted in an ADD catalog which provides means for customization such that for the project at hand only relevant decisions and decision details are provided to decision makers. This tailorability involves supporting visualizations (e.g. variability models such as feature diagrams; [17]) as well as auxiliary content: decision-making skeletons, prototypical designs, cross-references between related decisions, and frequency data on known uses.

VII. RELATED WORK

Besides the related work in the field of ADDs [3], [7]–[12], which provides the motivation for our work and which was already discussed throughout Sections I–III, our approach is mainly related to approaches in the broader software engineering context adopting frequent item-set analyses and association-rule mining on items other than ADDs. In these approaches, units of code (e.g., files, classes, commits) and of architectural design (e.g., components) are the analysis items, typically extracted from artifact repositories (source-code management systems, source-code bases). It follows naturally that these approaches do not directly compare with our distillation approach. The point of conceptual reference [31] and algorithmic foundations (`Apriori` or its variants such as DOAR) are, however, shared by virtually all related work.

Software clustering [14], [15]: Clustering code entities in software using association rules to recover architectural views to facilitate program-understanding tasks has been repeatedly reported (see [15] for an overview). [32] proposed item-set mining for establishing patterns of tangling in (COBOL) programs via files. An item set represents a single program, with each item representing a file used by the program. Based on a "grouping table" (actually a matrix relating supported-ordered programs and files), groups of programs sharing files of a certain support were manually identified. This represents an ad hoc form of frequent item-set analysis. [33] explore association-rule mining (rather than item sets) for software clustering, i.e., grouping software block entities (COBOL procedures) into subsystems to facilitate program comprehension and maintenance tasks based on shared characteristics. To decompose unstructured legacy system artifacts (files, functions, and data types) into modules, [34] employ a frequent item-set analysis as an integrated step of an architecture-recovery approach based on the Architectural Query Language (AQL). The computed item sets serve both for an automated modularization procedure and as a query optimization (closeness score). [35] present a requirements-driven approach (ArchMine) for recovering architectural entities (represented as UML packages) and their interactions (UML sequences) from object-oriented software systems based on co-occurrence patterns (of classes) in execution traces. The execution traces are obtained from instrumenting use-case scenario executions.

Change-impact analysis: [36] apply association-rule mining to detect and encode file-based change patterns in SCM (CVS) histories as association rules. Change patterns of interest are sets of co-changing source-code files for the scope of typical SCM artifacts (e.g., commits, change sets). Based on a knowledge base of such frequently co-changing file sets, a developer can obtain recommendations of files potentially affected by a change on a starting file (e.g., that she is currently working on). [37] equally devise associaton-rule mining for software configuration management systems (CVS) stressing change prediction, avoidance of incomplete changes, and detecting artifact coupling (co-changes) beyond source code (e.g., code and documentation). Again, atomic change sets are derived from a CVS repositories by grouping per-file changes based on fixed time boxes. [38] adopt association-rule mining for co-change prediction as part of the CLIO approach for detecting modularity violations in evolving (Java) software systems.

Conformance checking [16]: [39] mine association rules from abstracted inter-class dependencies along the revision history of (Java) systems. The rules are meant to guide conformance checking (absences, divergences) of the evolving architecture against an architectural reflection model.

VIII. CONCLUSION

In this paper, we have addressed the problem to date that the process of distilling architectural design decisions and their relationships remains mostly an informal, ad hoc process. The levels at which design spaces are systematically organized

and prioritized invite improvement. Our approach combines various methods of data collection, open and guided coding techniques, and frequent item-set analysis to distill decisions and decision relationships systematically. This is immediately useful to structure and to analyze a design space during decision identification, and helps prioritize decisions and decision details for decision makers regarding their relevance for a project at hand. The so far ad hoc, informal process is replaced by a structured process, and even though there are limitations due to the qualitative nature of methods of data collection and/or coding, most of the existing limitations can be mitigated to a large extent (see Section VI). We have demonstrated the practical applicability of our approach using a larger study on designs and architectures of 80 UML-based DSMLs. As future work, we will enable our approach to distill and to preserve the ordering of decisions. This requires adjustments at the level of data-collection techniques and the frequent item-set analysis.

REFERENCES

[1] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *Proc. 5th Working IEEE/IFIP Conf. Softw. Archit. (WICSA'05)*. IEEE, 2005, pp. 109–120.

[2] H. van Vliet and A. Tang, "Decision making in software architecture," *J. Syst. Softw.*, 2016, in press.

[3] N. Harrison, P. Avgeriou, and U. Zdun, "Using patterns to capture architectural decisions," *IEEE Softw.*, vol. 24, no. 4, pp. 38–45, 2007.

[4] M. Shahin, P. Liang, and M. R. Khayyambashi, "Architectural design decision: Existing models and tools," in *Joint Proc. 3rd Europ. Conf. Softw. Archit. and 8th Working IEEE/IFIP Conf. Softw. Archit. (WICSA/ECSA'09)*. IEEE, 2009, pp. 293–296.

[5] D. Falessi, G. Cantone, R. Kazman, and P. Kruchten, "Decision-making techniques for software architecture design: A comparative survey," *ACM Comput. Surv.*, vol. 43, no. 4, pp. 33:1–33:28, 2011.

[6] D. Tofan, M. Galster, P. Avgeriou, and W. Schuitema, "Past and future of software architectural decisions: A systematic mapping study," *Inform. Softw. Tech.*, vol. 56, no. 8, pp. 850–872, 2014.

[7] O. Zimmermann, T. Gschwind, J. Küster, F. Leymann, and N. Schuster, "Reusable architectural decision models for enterprise application development," in *Proc. 3rd Int. Conf. Quality Softw. Archit. (QoSA'07)*, ser. LNCS, vol. 4880. Springer, 2007, pp. 15–32.

[8] O. Zimmermann, J. Koehler, F. Leymann, R. Polley, and N. Schuster, "Managing architectural decision models with dependency relations, integrity constraints, and production rules," *J. Syst. Softw.*, vol. 82, no. 8, pp. 1249–1267, 2009.

[9] O. Zimmermann, U. Zdun, T. Gschwind, and F. Leymann, "Combining pattern languages and reusable architectural decision models into a comprehensive and comprehensible design method," in *Proc. 7th Working IEEE/IFIP Conf. Softw. Archit. (WICSA'08)*. IEEE CS, 2008, pp. 157–166.

[10] A. Jansen, J. Bosch, and P. Avgeriou, "Documenting after the fact: Recovering architectural design decisions," *J. Syst. Softw.*, vol. 81, no. 4, pp. 536–557, 2008.

[11] I. Lytra, S. Sobernig, and U. Zdun, "Architectural decision making for service-based platform integration: A qualitative multi-method study," in *Joint Proc. 10th Working IEEE/IFIP Conf. Softw. Archit. & 6th Europ. Conf. Softw. Archit. (WICSA/ECSA'12)*. IEEE CS, 2012, pp. 111–120.

[12] I. Lytra, H. Tran, and U. Zdun, "Supporting consistency between architectural design decisions and component models through reusable architectural knowledge transformations," in *Proc. 7th Europ. Conf. Softw. Archit. (ECSA'13)*, ser. LNCS, vol. 7957. Springer, 2013, pp. 224–239.

[13] S. Sobernig, B. Hoisl, and M. Strembeck, "Extracting reusable design decisions for UML-based domain-specific languages: A multi-method study," *J. Syst. Softw.*, vol. 113, pp. 140–172, 2016.

[14] A. van Deursen, C. Hofmeister, R. Koschke, L. Moonen, and C. Riva, "Symphony: view-driven software architecture reconstruction," in *Proc. 4th Working IEEE/IFIP Conf. Softw. Archit. (WICSA'04)*. IEEE, 2004, pp. 122–132.

[15] O. Maqbool and H. Babri, "Hierarchical clustering for software architecture recovery," *IEEE Trans. Softw. Eng.*, vol. 33, no. 11, pp. 759–780, 2007.

[16] L. T. Passos, R. Terra, M. T. Valente, R. Diniz, and N. C. Mendonça, "Static architecture-conformance checking: An illustrative overview," *IEEE Softw.*, vol. 27, no. 5, pp. 82–89, 2010.

[17] P. Kruchten, P. Lago, and H. van Vliet, "Building up and reasoning about architectural knowledge," in *Proc. 2nd Int. Conf. Quality Softw. Archit. (QoSA'06)*, ser. LNCS, vol. 4214. Springer, 2006, pp. 43–58.

[18] U. Heesch, P. Avgeriou, and R. Hilliard, "A documentation framework for architecture decisions," *J. Syst. Softw.*, vol. 85, no. 4, pp. 795–820, 2012.

[19] C. Borgelt, "Frequent item set mining," *Wiley Int. Rev. Data Min. and Knowl. Disc.*, vol. 2, no. 6, pp. 437–456, 2012.

[20] M. Hahsler, B. Grün, and K. Hornik, "arules—a computational environment for mining association rules and frequent item sets," *J. Stat. Softw.*, vol. 14, no. 15, pp. 1–25, 2005.

[21] J. Singer, S. E. Sim, and T. C. Lethbridge, "Software engineering data collection for field studies," in *Guide to Adv. Empir. Softw. Eng.* Springer, 2008, pp. 9–34.

[22] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, "Requirements engineering paper classification and evaluation criteria: A proposal and a discussion," *Requir. Eng.*, vol. 11, no. 1, pp. 102–107, 2005.

[23] M. Schreier, *Qualitative Content Analysis in Practice*. Sage, 2013.

[24] J. Saldaña, *The Coding Manual for Qualitative Researchers*, 2nd ed. Sage, 2013.

[25] M. Strembeck and U. Zdun, "An approach for the systematic development of domain-specific languages," *Softw. Pract. Exper.*, vol. 39, no. 15, pp. 1253–1292, 2009.

[26] I. Rival, "The diagram," in *Graphs and Order*, ser. NATO ASI Series. Springer, 1985, vol. 147, pp. 103–133.

[27] P. Mayer, A. Schroeder, and N. Koch, "MDD4SOA: Model-driven service orchestration," in *Proc. 12th Int. Conf. Enterp. Distrib. Object Comput. (EDOC'08)*. IEEE, 2008, pp. 203–212.

[28] Y. Bastide, N. Pasquier, R. Taouil, G. Stumme, and L. Lakhal, "Mining minimal non-redundant association rules using frequent closed itemsets," in *Proc. 1st Int. Conf. Computational Logic (CL'00)*, ser. LNCS, vol. 1861. Springer, 2000, pp. 972–986.

[29] F. Buschmann, K. Henney, and D. C. Schmidt, *Pattern-oriented Software Architecture – On Patterns and Pattern Languages*. John Wiley & Sons, 2007.

[30] D. Falessi, L. C. Briand, G. Cantone, R. Capilla, and P. Kruchten, "The value of design rationale information," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 3, pp. 21:1–21:32, Jul. 2013.

[31] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Proc. 1993 ACM SIGMOD Int. Conf. Management Data (SIGMOD'93)*. ACM, 1993, pp. 207–216.

[32] C. M. de Oca and D. L. Carver, "Identification of data cohesive subsystems using data mining techniques," in *Proc. Int. Conf. Softw. Maintenance (ICSM'98)*. IEEE, 1998, pp. 16–23.

[33] C. Tjortjis, L. Sinos, and P. Layzell, "Facilitating program comprehension by mining association rules from source code," in *Proc. 11th IEEE Int. Worksh. Program Comprehension (IWPC'03)*. IEEE CS, 2003, pp. 125–132.

[34] K. Sartipi, K. Kontogiannis, and F. Mavaddat, "Architectural design recovery using data mining techniques," in *Proc. 4th Europ. Conf. Softw. Maintenance Reeng. (CSMR'00)*. IEEE, 2000, pp. 129–139.

[35] A. Vasconcelos and C. Werner, "Evaluating reuse and program understanding in ArchMine architecture recovery approach," *Inform. Sciences*, vol. 181, no. 13, pp. 2761–2786, 2011.

[36] A. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE Trans. Softw. Eng.*, vol. 30, no. 9, pp. 574–586, 2004.

[37] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in *Proc. 26th Int. Conf. Softw. Eng. (ICSE'04)*. IEEE CS, 2004, pp. 563–572.

[38] S. Wong, Y. Cai, M. Kim, and M. Dalton, "Detecting software modularity violations," in *Proc. 33rd Int. Conf. Softw. Eng. (ICSE'11)*. IEEE, 2011, pp. 411–420.

[39] C. A. Maffort, M. T. Valente, R. da Silva Bigonha, A. Hora, N. Anquetil, and J. Menezes, "Mining architectural patterns using association rules," in *Proc. 25th Int. Conf. Softw. Eng. Knowledge Eng. (SEKE'13)*. Knowledge Systems Institute Graduate School, 2013, pp. 375–380.