Customized Choreography and Requirement Template Models as a Means for Addressing Software Architects' Challenges

Nebojša Taušan, Sanja Aaramaa, Pasi Kuvaja,
Jouni Markkula, Markku Oivo
Department of Information Processing Sciences
University of Oulu
Oulu, Finland
{nebojsa.tausan, sanja.aaramaa, pasi.kuvaja,
jouni.markkula, markku.oivo}@oulu.fi

Jari Lehto
Segment Manager
Nokia Networks
Espoo, Finland
jari.lehto@nsn.com

***Abstract*—Software architecture designs are useful artifacts; however, their development and maintenance are considered challenging. To better understand the possible causes for these challenges, this article presents a case-study intended to discover and understand software architects' challenges and to propose domain-specific models to address these challenges. The main results of the case-study include a) the classification of challenges in software architecture design as well as an interpretation of the rationale behind these challenges, and b) two domain-specific models for addressing architects' challenges through architectural design. The proposed models are expected to facilitate communication between development teams, and to improve the technical aspects of the information content of requirements.**

Keywords- Software Architecture; Case-study; Choreography; Requirements Engineering; Challenge.

I. INTRODUCTION

Throughout the software product life cycle, well-established Software Architecture (SA) design is considered a valuable asset that can guarantee several quality aspects, as well as efficient development and maintenance work [1]. Today, software architects have a substantial amount of knowledge and a plethora of methods and tools at their disposable; still, well-established SA designs are scarce. One of the reasons for this situation is that, according to Falessi et al. [2], there is no SA design methodology that can simultaneously meet all the needs of an architect. In this study, the assumption is that the growing complexities of SA design challenges are one of the main reasons for the scarceness of well-established SA designs. The plethora of challenges that architects face during their work is reported in several empirical studies. Some of these studies are presented in more detail in the following paragraphs.

Smolander and Päivärinta [3] analyzed stakeholders participating in SA design and reported their problems in relation to SA. The problems, or challenges, that were expressed by software architects included: a) the continuous lack of skilled architects, which resulted in a need for well-documented SA specifications, and b) the communication mismatch, which results from architects' need to communicate with other stakeholders who often lack the necessary technical knowledge and insights.

In [4], Bosch presents his view on SA design challenges along with proposals for how to overcome them. These challenges include the lack of first-class representation, cross-cutting and intertwined design decisions, high costs of change, design rules and constraints violations, and obsolete design decisions failing to be removed from SA designs.

The challenge of enriching existing software development practices with architectural thinking is reported by Lattanze in [5]. Besides the conclusion that common methods of disseminating architectural knowledge do not work, the author proposes a list of challenges that lead to challenge state. Among others, the list includes the lack of resources for SA design, the ill-treatment of architecture activities, lack of career path for architects, and the fact that created SA designs are not used.

One of the promising ways to overcome architects' development challenges is the utilization of a Model-Driven Engineering approach [6]. In short, this approach allows architects to identify the areas in SA design that they see as particularly challenging and express these areas with Domain-Specific Models (DSM). The identified areas are then specified and managed using the concepts, rules and relationships defined in the DSM. The utilization of the domain-specific approach for the specifications and management is expected to yield several benefits, such as better comprehension of specifications, faster development and enhanced productivity [7][8][9]. The Model-Driven Engineering approach represents the overall context of this study.

To better understand and learn about SA design challenges in a real-life setting, a case-study with four software development companies was conducted. The main results are presented in this article. The main study goals were to identify a software architect's challenges and to propose DSMs as a means to address those challenges. Stated goals were reached by answering to the following research questions:

- *RQ1: What challenges do software architects face during the development and maintenance of software architecture design?*
- *RQ2: How to address the identified challenges with domain-specific models?*

These research questions were answered by conducting and analyzing five interviews with software architects,

 ISBN: 978-1-61208-367-4

analyzing additional interviews from previous studies, consulting the relevant literature, analyzing company-specific documentation, and closely collaborating with industry experts.

The stated case-study goals are also aligned with the goals of the AMALTHEA project. AMALTHEA is a European ITEA2 project of which this study is a part of, and its main goals include the development of an open source tool integration platform, the creation of an engineering methodology, and the specification of a tool-chain that will support all relevant software development areas with methods and DSMs [10]. The case-study results support AMALTHEA's goals by identifying the challenges faced by software architects on the basis of which the DSMs will be proposed. Proposed DSMs will serve as a foundation for the development of distinct tools which will become a part of AMALTHEA tool-chain.

The structure of this article consists of six sections. The following section, Section II, introduces the research method. This section is followed by the research results, which are described in Section III and Section IV. A validity discussion is presented in Section V. Concluding remarks and future research directions are outlined in Section VI.

II. RESEARCH METHOD

In this study, software architects, their challenges and model proposals are studied in their natural context. Accordingly, the case-study approach was selected as an overall research approach [11]. The research activities within the case-study were divided into two major phases, each of which sought to provide the answer to one research question. In the first phase, the SA design challenges were identified, categorized and interpreted based on knowledge gathered through an interview of the company experts. In the second phase, new DSMs were developed in such a way as to address the identified challenges. The knowledge resulting from the first phase represented the inputs to the activities in second phase. The two phases of the case-study, labeled as Phases A and B, together with the corresponding topics under investigation, the relationships between those topics, and RQs they answer, are presented in Figure 1. The research activities undertaken in these phases are described in more detail in the subsections bellow.

A. Research Phase A

The main purpose of Phase A was to provide the knowledge necessary for the development of DSM proposals. Since the DSMs seek to address the challenges faced by architects, the knowledge here implies concrete challenges, which were categorized and interpreted. For this purpose, the researchers adapted the thematic analysis method following Miles and Huberman's guidelines [12]. The main reason that a qualitative method was selected for this phase is that such a method provides a useful starting point for studying phenomena for which existing knowledge is scarce [13]. SA design challenges can be seen as such a phenomenon. The adaption of the thematic analysis will be presented through the two major phases: data collection and data analysis.

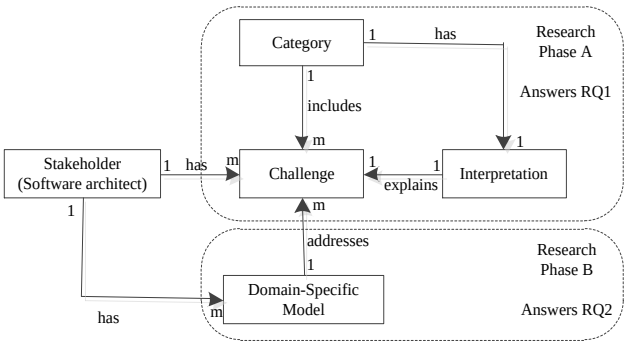

Figure 1. Case study overview.

Data collection: According to Falessi et al. [14], empirical methods, such as interviews, are suitable data collection techniques for studying SA. Following this recommendation, the authors used five interviews as the primary source of information for this study. The interviews were conducted during the first quarter of 2012, with interviewees who were working in the role of a software architect, and who had between 10 and 26 years of experience in software development.

The interviews were conducted as semi-structured, which allowed researchers to define the themes of interest, but also allowed interviewees to express their views regarding these themes in the way that was most suitable for them. Broad themes covered by the interview questions included interviewees' backgrounds, their understanding of what SA is, things that are seen as challenges and things that are seen as improvements. Additional data about the interviews are included in Table I.

In addition to the interview data, the large ICT company with which the authors collaborated provided company-specific documentation related to technical analysis. This documentation included: templates, process and work descriptions, example requirements and test specifications. This documentation was mostly used in Phase B, during the development of models, but it was also used as a means to better understand the interview response and to put these responses in context. For the purpose of data triangulation, supplementary interviews from a previous study [15] were utilized as well. Relevant information about interviewees from these supplementary interviews is presented in Table I.

To ensure the accuracy and the high quality of the data, the following measures were taken: a) The questionnaire used for the data collection was developed by a single researcher, but reviewed by at least two senior researchers and one industry expert. This was also the case for the supplementary interviews used during the study. b) The interviews were recorded, transcribed, and sent to

TABLE I. INTERVIEW DATA

Company	Type	Country	Method	Duration
A	Large ICT	A	Telephone call	1 h
A	Large ICT	A	Face to face	1.5 h
B	SME ICT	A	Telephone call	1 h
C	SME ICT	B	Telephone call	1.5 h
D	Consultant	A	Face to face	2 h
Supplementary interviews				
A	Large ICT	C	Telephone call	1 h
A	Large ICT	D	Telephone call	1.5 h

 ISBN: 978-1-61208-367-4

the interviewees for verification and for the clarification of terms that were unclear to the researchers. Upon finalization of the analysis, the results were sent for verification to industry experts in the form of technical reports and were presented in the workshops. c) Researchers worked under non-disclosure agreements and the project consortium agreement, which protected the privacy of the interviewees.

Data analysis: To aid the analysis, interview transcripts and company-specific data were imported into the NVivo tool [16], which is a software package for qualitative data analysis. A distinctive feature of this tool allowed the researchers to work on the same data sources and to continuously have insight into one another's work. This feature was especially useful because it allowed for mutual verification of work "on the fly".

At the core of the thematic analysis approach is the technique of coding. Coding allows a researcher to relate pieces of text that are of interest to the analysis with specific names or *codes*. The subsequent analysis of the text under each code facilitates the development of themes (i.e., categories) and for the rendering of interpretations. Code and category development, as well as their interpretations, are used to structure the explanation of the data analysis.

Code development: First, every piece of text that interviewees explicitly mentioned as a challenge, as well as text, that based on the researchers' expertise was known to be a challenging aspect of SA design, was encoded. The pieces of text under each code helped researchers gain a deeper understanding of SA-related problems and to formulate these problems as the challenges presented in this article. These challenges are the foundational concept of this study since they represent the basis for the development of DSMs (cf. Figure 1).

Category development: Newly formulated challenges were expressed as new codes. In the following iteration, the interview transcripts were re-coded using these new codes. The coded text was further analyzed to find commonalities, and in this case, four themes reflecting the underlying causes for the identified challenges were proposed. These themes, or categories, were used to organize the challenges and to facilitate their interpretation.

Interpretation: The final step in the data analysis was interpretation, in which the researchers combined and summarized what had been learned from the interviews with their own existing knowledge and experience. The main goal of this step was to go beyond the challenges and categories, to add the explanations and rationales behind these challenges.

The challenges, categories, interpretations, and relationships between them are illustrated in Figure 1, and, together, they represent the core knowledge necessary for the development of DSM proposals.

B. Research Phase B

Research Phase B used the results from the previous research phase for the development of DSM proposals. For this purpose, a number of workshops were organized in which industry experts, together with researchers, analyzed the challenges, categories, and their interpretations. During these workshops, challenging areas for which DSMs could be developed were identified. The first such area was described as the lack of system-level agreement on responsibilities during the implementation phase, while the second area was identified as the lack of adequate technical information in the requirement document.

Once these areas were identified, the researchers consulted the relevant literature and used company-specific materials and their own expertise to structure proposals for addressing the challenges through DSM. For the first identified area, a choreography-based DSM was proposed, while, for the second, researchers proposed a DSM for the dynamic requirement template. These two proposals were developed for the context of the case company which develops large embedded software systems and, therefore, were strongly influenced by the case company's practice. Still, the ideas within proposals are considered generic enough to be useful to architects in other companies as well.

The way in which the developed DSMs relate to the previous research phase is illustrated in Figure 1, while the more elaborate explanations of research results (i.e., challenges, categories, interpretations, and DSMs), are presented in the following two sections.

III. SOFTWARE ARCHITECTS' CHALLENGES

In this section, the results of the research Phase A are presented. These results were obtained using interview data and the thematic analysis approach, and they include the identified challenges, categories, and interpretations. Here, the derived categories are used to organize the presentation of concrete challenges and their corresponding interpretations.

A. Challenges, categorization and interpretation

The identified challenges are organized into four categories: knowledge, global software development, system size and complexity, and architectural viewpoints. This categorization seeks to reflect the underlying causes for the identified challenges.

Knowledge category: The development of SA designs, or architecting, is a knowledge-intensive process. Large amounts of both theoretical and practical knowledge are required to fulfill daily tasks. The analysis of the collected data revealed five challenges whose causes can be traced to the lack of knowledge. These challenges are summarized in Table II, and their interpretation is presented in the text below.

TABLE II. KNOWLEDGE RELATED CHALLENGES

ID	Challenge
K1	Architecting is usually experience based, without any clear statement about the rationales for design constructs or decisions.
K2	Architecting is done in the uncertain conditions. Needed information is missing.
K3	Architecting is done in the uncertain conditions Needed information is not reliable.
K4	Software architect replacement.
K5	Communicating the architecture between the developers.

K1: SA theory and SA design techniques are not sufficiently included in the educational background of software architects. Consequently, each architect devises his or her own personal understanding about SA concepts and practices and uses this understanding to specify the underlying logic behind SA designs. Since these design specifications are heavily burdened with architects' personal experiences and understandings, communicating designs to other architects becomes a challenge.

K2: Architects often do not receive the information necessary for their work. This leads to additional time consumption for information gathering and the usage of informal communication channels. What is discussed and agreed during informal communication can be important for understanding certain architectural solutions, but it often remains undocumented and can be forgotten.

K3: Two explanations for this challenge are possible: a) differences in education and experience can cause misunderstandings, and b) large systems are often documented from specific points of view. What is meaningful from one viewpoint can be irrelevant from another.

K4: During their work, architects gain knowledge about systems, interdependencies, processes, people, and customers, and they use this knowledge to develop SA designs. In some cases, architects are displaced during the course of development. The work done by a displaced architect is often poorly documented and experience based (see also K1), and for these reasons it takes a significant amount of time to train the novice architect who will continue the work of the outgoing architect.

K5: Employees often have different understandings about the same concepts. Terms like component, domain, and functional area are defined in the literature, but they are often interpreted differently by practitioners or used differently in different contexts. Refer also to challenges K1 and K3.

Global software development category: Software development companies often operate across several locations worldwide. In such a development setting, project teams are formed with developers coming from various cultural backgrounds and time zones and who communicate using non-native languages. Our analysis revealed four problems that can be linked to such a development setting (cf. Table III).

G1: Two explanations for this challenge are possible: a) For most team members, working in global development setting means communicating in a non-native language. Communicating complex issues requires a high level of language proficiency, which does not always exist. b) Global communication is done via different tools, such as emails, faxes, Wikis and voice calls. These means are not necessarily considered good substitutes for face-to-face communication.

G2: Due to mergers and acquisitions, companies are faced with the task of imposing different rules and practices. For example, if one company uses agile development, while another uses a traditional development approach, employees will be obligated to accept a new way of working.

G3: Personal acquaintances and face-to-face communication is highly appreciated among architects, and often seen as the best method of problem solving. However, this type of communication in global software development setting requires a substantial amount of resources; therefore, it always has to be justified in terms of the costs and benefits that will accompany it.

G4: Due to the variety of tasks and the large number of teams that are scattered throughout the globe, the precise responsibilities of architects are not always clear.

System size and complexity category: The interviewees work with software systems that are considered large and complex. The phrase "large and complex" emphasizes the variety of different implementation technologies, software platforms, development teams and features that such systems support. Size and complexity cause a number of challenges. The interview analysis revealed six of these challenges, which are presented in Table IV.

S1: The development of an architecture for large software systems is hampered by frequent changes, such as a) changes in organization (similar to G2), b) changes in, for example, requirement and feature documents, c) changes in release content, and d) changes in technology.

S2: Different teams prefer different practices and technologies. Sometimes, these technologies are mutually exclusive, and in these circumstances architects must decide in favor of one technological solution.

S3: System functionality can often be implemented in different architectural parts. A consensus must be reached among architects regarding which functionality will be allocated to which architectural part. This is especially important in cases for which various architectural parts are also distinct sellable items. Allocating functionality in one architectural part, means making that part a more lucrative investment option for customers.

S4: Large systems have a large number of stakeholders. Each stakeholder has his or her own vision for how the system should work, which is expressed through specific requirements. Often these requirements conflict with one another, and it is up to the architects to decide how to reconcile these conflicts.

S5: Systems tend to become large, while architects tend to become focused only on distinct parts. This state results in a loss of understanding about systems "as a whole". Systems are only valuable as a "systems" - that is, as a whole. If several parts are preforming well, but other parts are creating

TABLE III. GLOBAL SOFTWARE DEVELOPMENT CHALLENGES

ID	Challenge
G1	Difficulties in communicating tasks and results.
G2	Merging different architecting practices.
G3	Lack of personal acquaintances and face-to-face communication.
G4	Architects' responsibilities are not clear.

TABLE IV. SIZE AND COMPLEXITI CHALLENGES

ID	Challenge
S1	Architecting in a changing environment.
S2	Architecting in a heterogeneous environment.
S3	Architecting in a competitive environment.
S4	Architecting in a conflicting environment.
S5	Narrowly focused architecting.
S6	Models and tools are not sufficient for current architecting needs.
S7	Architecture and implementation often (mis)align.

 ISBN: 978-1-61208-367-4

bottlenecks, the overall system's performance becomes questionable. The performance of all parts must be balanced and planned - so that the overall performance is optimized.

S6: Conventional modeling techniques and tools are not sufficient for architects' needs. For example, the model or format of a requirement can be sufficient for one group of stakeholders, but insufficient for another. Different groups, working on different problems, have different expectations for models and tools.

S7: Large systems have large architectures that must be followed by developers. However, there are no means by which to verify that, for example, the source code for the release actually follows the architecture. Since new releases tend to reuse old designs, this misalignment can result in huge losses in time and resources.

Architectural viewpoints category: Viewpoints represent one of the crucial concepts for documenting software architecture. Architecture is actually expressed as a collection of views [17][18] based on several viewpoints. Each viewpoint emphasizes elements, and provides data that are significant only for specific concern(s) tied to a particular viewpoint. Other elements and data are omitted for clarity reasons. Based on their needs, architects can develop a feature viewpoint, a component viewpoint, a performance viewpoint, a maintenance viewpoint, and many others viewpoints they find useful. However, besides benefits, the existence of different viewpoints also causes challenges (cf. Table V).

V1: Each viewpoint represents a "world" for itself. It has its own purpose terminology, conceptualization and rules which must be known and understood in order to be effectively used, discussed and decided. Sometimes employees discuss things from the perspective of different viewpoints. This can lead to communication problems, which hamper the development process.

V2: A viewpoint addresses certain concern(s), but it does not exist in isolation. Typically, viewpoints rely on each other, meaning that updating one viewpoint often requires updating and validating other viewpoints as well. These relationships are often neither explicit, nor maintained.

V3: A reference architecture is an artifact whose purpose is to be shared across all development teams. It represents a common vision, or a shared mental model that sets common rules and terminology. The system described from this particular viewpoint is often seen as a reference for communication and development. The study revealed, however, that the reference architecture is not always properly maintained.

V4: Architectural designs, or views, are not used to their full potential. Often only a small portion of a design is used, while the rest of the information it offers remains neglected.

TABLE V. CHALLENGES RELATED TO VIEWPOINTS

ID	Challenge
V1	Employs are not aware of the existence of different viewpoints
V2	Relationship between viewpoints is not clearly visible
V3	No common, comprehensive reference architecture
V4	Architectural designs (views) are used too narrowly
V5	Architectural designs (views) are misused
V6	Difficulties in enforcing viewpoints

V5: Development problems are often discussed from only one viewpoint, and, as a result, wrong design decisions are made. For example, a static structure can be useful for an efficient breakdown of work, but it would be risky to use such a structure as a solution for certain other problems such as requirements breakdowns.

V6: In order to reach its full potential, a viewpoint must be used and understood by all interested stakeholders. A company that operates worldwide may encounter problems in enforcing certain viewpoints or practices related to these viewpoints throughout all of their global departments (refer also to G1).

IV. DOMAIN-SPECIFIC MODELS PROPOSAL

In this section, the results of the second research phase are presented. These results include two DSMs which were developed based on the identified challenges and which seek to address two subsets of those challenges. The structure for the presentation of the two models includes the following parts: a) context which explains the circumstances from which the challenges were identified; b) challenge area, which explains the architects' interest and identifies which of the identified challenges the model includes; c) proposal, which provides a description of the proposed DSM; and d) theory, which presents a short overview of the theoretical foundations underpinning the proposed DSM.

Both DSM proposals share a common underlying assumption, which is that there is an interrelationship between the product breakdown and the way in which development teams are organized. The logic of the "product-team breakdown" assumption is known in software development and reported in, for example, [19]. A simplified version of this logic is illustrated in Figure 2.

A system as a whole is subdivided into several logical components, which are further subdivided into more fine grained logical components. These components are mapped into real, physical software components, which are illustrated as the leaves of the hierarchy on the left side of Figure 2. Software development teams are organized following the same hierarchical structure. As illustrated, the board of architects is responsible for the high level conceptualization of the overall system, which is then operationalized by architects and their development teams. Each development team is responsible for a dedicated logical component, and its corresponding physical components. With this assumption in mind, the following subchapters present the detailed explanations of the two proposals.

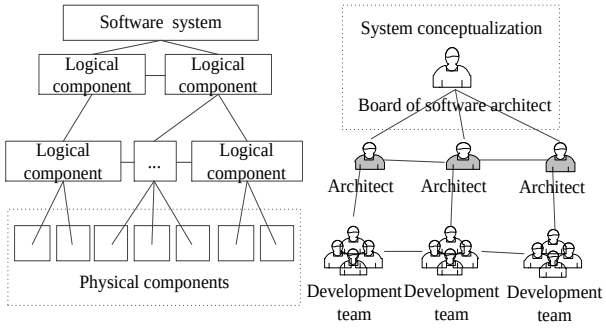

Figure 2. A system breakdown and team organization.

1) Proposal 1: Choreography based agreements

Context: The case company employs several hundred developers in its R&D division. The main task of those developers is to ensure the continuous evolution and maintenance of a large, embedded software system. The developers are organized in teams and, as illustrated in Figure 2, each team is responsible for a distinct, logical part of the system. Due to the large number of teams which are typically dispersed across different geographical, national, and cultural locations, developers are often unaware of their role in the "big picture". The "big picture" here denotes an understanding of how a developer's everyday work is aligned with the work of other teams and how it affects the functioning of a system as a whole.

Challenge area: There is no system-level agreement that would increase developers' awareness regarding who does what, and in which order. This leads to work duplication, reworks, frequent delays, and a loss of opportunities from the parallelization of work. The problem of duplication of work, for example, is explicitly stated by one of the interviewees:

"Truly, there is not such a company-level function where a decision could be made that a specific solution is implemented in a specific product and not in some other product. In practice, there may be several products that provide technical solution for system level need, and, in addition, all the solutions are standardized."

This challenge area can be seen as a collection of several of the challenges faced by architects' which have been previously identified. These challenges are K5, S3, S5, S6 and, partially, G1. An explanation of the proposal and the rationale for why it can be seen as a potential solution to these challenges is given in the text below.

Proposal: A choreography model is a way to intervene in the challenge area. The proposal is to select, customize, and provide tool support for the choreography modeling, by supplementing it with domain-specific content, and by merging it with additional models. Initial work on domain-specific supplements is begun, and some of the results are explained in Taušan et al. [20], where the way how different implementation of middleware features are affecting the choreography model is studied.

The goal behind the merger of choreography and other models is to create more ways for architects to express their designs. For example, the WS Choreography model [21] prescribes constructs for representing, e.g., the interaction. A merger provides an additional option to express interactions using techniques such as UML state charts, or UML-collaborations.

Theory: Choreography represents a system-level view of the interactions between distinct system parts [22]. The semantics of a choreography model allow architects to capture and analyze the use case in terms of participants, their roles, their messages, and the order in which those messages are exchanged in order to fulfill the use case [23]. Referring to Figure 2, each participant represents a distinct development team or engineering unit within the company. The role indicates the contribution of the architectural part, which is embodied in physical components under the team's responsibility. Messages and message ordering have to do with what is exchanged between the roles, as well as when the exchange occurs. The simplified illustration of the choreography model instance is presented in Figure 3. Here, four teams (teams x, y, z, and q) are participating in fulfilling the use-case, while the components under their responsibility take six roles (roles A, B, C, D, E and F).

The semantics of the choreography model, the experiences published in literature, and the possibilities for customizations were the main arguments for proposing it as a potential solution for the challenges in the challenge area. These arguments are discussed in more detail below.

The challenge of communicating the SA (ID: K5) is explained through the ambiguity and misunderstanding of the concepts in use. One way to address this challenge is to customize the choreography model by including domain-specific concepts. The rationale behind this approach involves reported evidence that the inclusion of domain-specific concepts can improve the comprehension and readability of specifications [7][8], which are at the core of this challenge. Moreover, this approach partially addresses the challenge of communicating tasks and results (ID: G1).

The challenge related to competing environments (ID: S3) involves allocating functionality to a set of architectural parts. Choreography natively supports the role concept for documenting the contribution that an architectural part provides to the fulfillment of the use-case. In the proposed approach, the focus is on the role, as a means of addressing this challenge, by providing the methodological and tool support for role identification and management. The rationale for using the role to understand the contribution of architectural parts at the analysis level, and to relate this to physical components during the implementation, is claimed to be a good practice by Kruger [24] and by Kruger, Nelson and Venkatesh [25].

The challenge of the narrow focus (ID: S5) involves comprehending the system as a whole and ensuring its performance. The reason choreography is seen as a suitable approach for this challenge is that it natively captures the interactions needed for the system-level use cases. As such, it imposes and documents the collaboration of all interested teams and provides insights into the roles that each team has. Regarding performance issues, the existing literature offers evidence that organizing systems according to a choreography model can result in better performance [26][27].

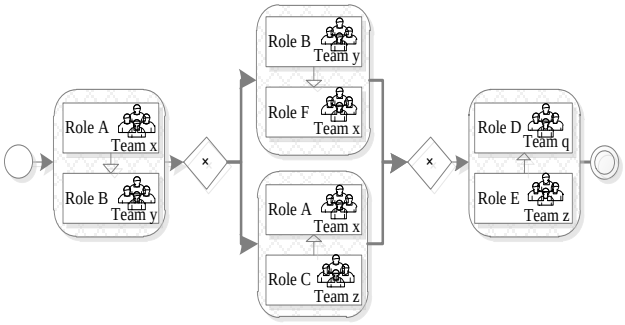

Figure 3. Choreography model.

The challenge of inadequate models (ID: S6) will be addressed through the merger of a choreography model and other models which are used by industry partners. Allowing architects to use their own preferred modeling techniques, together with the domain-specific constructs offered by the choreography model can be seen as an adequate response to this challenge.

2) Proposal 2: Dynamic requirement template

Context: In the case company, requirements are elicited by customer teams and then communicated to product management. At first, an initial screening is undertaken to quickly determine whether a requirement has the potential to bring value to the customer. If a value is identified, the requirement is analyzed in more detail from business and technical feasibility points of view (see Aaramaa et al. [15] for more details about such an analysis). This particular proposal improves the information content that is needed for the technical feasibility analysis.

Challenge area: Collecting the needed requirement information from customers and communicating this information to product management, and then to software architects, is the task of customer teams. The template for collecting and recording requirements, however, lacks the necessary technical information, and the reliability of the information in the requirement specifications is questionable. In addition, distinct technical information content has to be provided to describe each architectural part.

The direct consequence of this challenge is that architects use a significant portion of their time trying to find the necessary information, before they can begin the technical feasibility study and implementation of the requirement. This inefficient use of architects' time is only one example of the issues that are prevalent in this area, and it is also recognized by one of the interviewees:

"And because they [customer teams] *are technically not that well-trained or they don't have that kind of deep knowledge about the new functionality, (…) and then we* [software architects] *always have to make new and new inquiries towards them, to go back to the customer in order to get more information."*

This challenge area can also be seen as a collection of several architects' challenges that have previously been identified. These challenges include K2 and K3, as well as, partially, S1, S5 and S6. An explanation of our proposal and the theory that supports it is presented in the text below.

Proposal: The dynamic requirement template consists of two parts: a common and a specific part. The common part is the same for all requirements and consists of data such as the requirement's ID, name, priority, and description. The specific part is tied to a distinct part of the system or to a logical component, as is shown in Figure 2, and it consists of data that are relevant only for that specific system part. The main idea here is to use the specific part of the template to allow architects and their teams to define the information content that is relevant to their work.

This model of a requirement template is illustrated in Figure 4. The architects and their teams define the information content which includes data that have to be collected from customers, the descriptions of those data, guidance how to collect them, and criteria for the collected data's completeness. This information content forms the specific part of the requirement template. When this is done, the model is ready for instantiation by customer teams.

There are three distinct steps that can be identified during the template instantiation: a) recording data from the common part, b) understanding which parts of the systems are affected by the requirement and c) recording the specific part of the requirement for the identified system part. When these steps are completed, the requirement specification can be passed to the architects for technical feasibility analysis and implementation. It is expected that, due to the provision of focused technical data, architects and developers can do their work more efficiently.

Theory: The model behind the dynamic requirement template proposal is motivated by the idea that SA has a strong influence on Requirement Engineering (RE), and that including SA-related items in a requirement specification may result in different benefits. Some of the studies supporting this idea are presented below.

One of the first publications to focus on this idea is the panel discussion presented in Shekaran et al. [28]. In this panel, participants expressed their views on how SA is present in RE and outlined expected benefits. These benefits included an understanding of the resistance to change; the consistency, comparability, and feasibility of the requirements; and the consideration of different design alternatives.

Ferrari et al. [29] conducted a controlled experiment to understand the impact of architectures on new system requirements. The authors claimed that by considering SA during RE (among other things), analysts could elicit 10% more architecturally relevant requirements, 10% more "important" requirements, 7% more crosscutting requirements, and more implementation and interoperability requirements.

According to Cervantes et al. [30], frameworks as SA concepts influence RE. Frameworks can impose constraints such, as testability and developer skills, or create new system requirements. The example of new requirements is the case when the utilization of a concrete technology demands the usage of a concrete application type. By considering this constraint early, (i.e., in RE), losses in later development phases can be avoided.

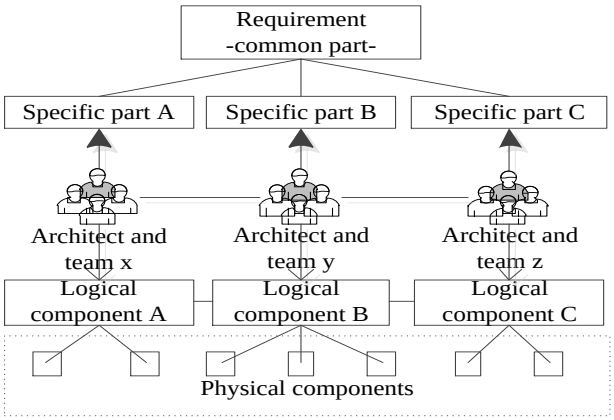

Figure 4. A dynamic requirement template.

V. VALIDITY DISCUSSION

The validity of a case-study, according to Yin [31], constitutes four aspects: construct validity, internal validity, external validity and reliability. There are several issues that may threat the validity aspects, and these were considered throughout the study.

If the researchers and the interviewees do not understand the concepts to be studied in the same way, a threat to construct validity is introduced. This threat was mitigated in this study through the rigorous peer review of the interview questionnaires that were used to collect the data for both the primary and the supplementary interviews.

The utilization of supplementary interviews can represent another threat to validity, since these data were collected for another purpose and, thus, must be considered as third-degree data [11]. Using this type of third-degree data, however, may also mitigate threats to validity, since such data's use triangulates the data; moreover in this particular study, the results of the additional interviews were in line with the primary set of data. Thus, the additional interviews addressed the validity threat to generalizability that resulted from the relatively low number of interviewees in the primary set.

The fact that the researchers have years of experience of research co-operation in the context of the case company also poses a threat to reliability in the form of researcher bias. To mitigate this threat, measures for ensuring data quality and correctness were taken. These were presented in Section II.

A threat to internal validity relates to possibilities to generalize the results and draw cause–relationship conclusions from those results. This case-study did not seek to analyze causal relationships, so, from that viewpoint internal validity has not been considered.

External validity concerns how much an analysis's results can be generalized, (i.e., used in other companies). The analysis results for this study were based on qualitative data from four companies, which develop different types of systems in different domains. The diversity of the interviewees suggests that categories and challenges could be identified in other contexts as well. The improvement proposals, however, were developed in cooperation with experts from a single company. Beyond the educated opinion that these proposals are applicable in similar type of companies or context, no other argument can be provided regarding external validity. Therefore, a threat to external validity remains.

VI. CONCLUSION AND FUTURE WORK

SA design is a solid approach to ensuring software quality and longevity. Its importance in software development is undoubtedly confirmed by one of the interviewees who, for example, claimed that:

"When we have it, [software architecture] *work comes much easier."*

The goals expressed in the AMALTHEA project, however, represent an additional empirical argument that SA design practices still need improvements. Consequently, this article presents our results from the study in AMALTHEA project which is conducted to improve the understanding of what architects perceive as challenging in their daily practice, as well as to develop ways to address these challenges with DSM.

The main results of this study are two DSM proposals. These DSMs were developed using the discovered challenges, the challenge categories (which were devised to reflect the underlying causes), and the interpretations of the challenges. In addition, existing literature, company-specific material and researcher' expertise were also used during the DSM development.

These results are also seen as answers to the research questions that where stated at the beginning of this paper. In short, based on the data analysis, RQ1 is answered by identifying, categorizing, and interpreting the architects' challenges. To answer RQ2, the researchers used the RQ1 answers and proposed two DSMs: namely, choreography-based agreements and the dynamic requirement template. These two proposals have yet to be validated. It should be also noted that, based on the identified challenges, additional DSMs could be derived as well. Which combination of challenges an architect sees as suitable for addressing through DSM is highly influenced by the architect's experience and the development context.

In addition to using these results, software architects can also recognize the derived categories and use them to predict possible challenges they will face if, for example, their company operates in a global software development setting, their product becomes large and complex, or multiple viewpoints are in use. It is also important to emphasize that the knowledge category can be seen as a pervasive category, which is present regardless of software size, complexity, the utilization of viewpoints or global software development settings. The list of challenges under each category can be seen as the concrete points that can either be addressed through an architect's choice of development practice, or serve as a means through which to raise architects' awareness about the particular challenge.

In future work, the two proposals will be fully customized to fit the case company's context. Customization will include various tasks, such as specifying of the information content for the dynamic requirement template, supplementing the choreography model with details that are relevant to the developers, and developing software support for the proposals. Additionally, the authors plan to conduct a series of evaluations with industry practitioners to validate and improve the two proposals.

ACKNOWLEDGMENTS

This study was supported by ITEA2 and TEKES. The authors would like to express their gratitude to the interviewees for their time and effort, and to the AMALTHEA partners for their cooperation. The authors are also grateful to J. Peltonen from the Tampere University of Technology for his valuable suggestions regarding the choreography model proposal.

REFERENCES

[1] L. Bass, P. Clements, and R. Kazman, Software architecture in practice. Addison-Wesley Professional, 2003.

[2] D. Falessi, G. Cantone, and P. Kruchten, “Do architecture design methods meet architects’ needs?,” The Working IEEE/IFIP Conference on Software Architecture (WICSA’07), 2007, pp. 5.

[3] K. Smolander and T. Päivärinta, “Describing and communicating software architecture in practice: observations on stakeholders and rationale,” In Advanced Information Systems Engineering, 2002, pp. 117–133.

[4] J. Bosch, “Software architecture: The next step,” in Software architecture, 2004, pp. 194–199.

[5] A. J. Lattanze, “Infusing Architectural Thinking into Organizations.,” IEEE Software, vol. 29, no. 1, 2012, pp. 19 – 22.

[6] D. C. Schmidt, “Model-driven engineering,” Computer, IEEE Computer Society, vol. 39, no. 2, 2006, pp. 25 – 31.

[7] T. Kosar, M. Mernik, and J. C. Carver, “Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments,” Empirical Software Engineering, vol. 17, no. 3, 2012, pp. 276–304.

[8] M. Völter, “Architecture as Language,” IEEE Softw., vol. 27, no. 2, 2010, pp. 56–64.

[9] A. Van Deursen, P. Klint, and J. Visser, “Domain-Specific Languages: An Annotated Bibliography,” Sigplan Notes, vol. 35, no. 6, 2000, pp. 26–36.

[10] “AMALTHEA,” 2014. [Online]. Available: http://www.amalthea-project.org/. [Accessed: 09-May-2014].

[11] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” Empirical Software Engineering, vol. 14, no. 2, 2008, pp. 131–164.

[12] M. B. Miles and A. M. Huberman, Qualitative data analysis: An expanded sourcebook. Sage, 1994.

[13] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, “Selecting empirical methods for software engineering research,” in Guide to advanced empirical software engineering, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. Springer, 2008, pp. 285–311.

[14] D. Falessi, M. A. Babar, G. Cantone, and P. Kruchten, “Applying empirical software engineering to software architecture: challenges and lessons learned,” Empirical Software Engineering, vol. 15, no. 3, 2010, pp. 250–276.

[15] S. Aaramaa, T. Kinnunen, J. Lehto, and N. Taušan, “Managing Constant Flow of Requirements: Screening Challenges in Very Large-Scale Requirements Engineering,” Product-Focused Software Process Improvement, 2013, pp. 123–137.

[16] “NVivo 10 research software for analysis and insight,” 2014. [Online]. Available: http://qsrinternational.com/products_nvivo.aspx. [Accessed: 09-May-2014].

[17] “42010-2011 - ISO/IEC/IEEE Systems and software engineering - Architecture description,” IEEE Computer Society. 2011.

[18] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, and R. Little, Documenting software architectures: views and beyond. Pearson Education, 2002.

[19] N. Nagappan and T. Ball, “Using Software Dependencies and Churn Metrics to Predict Field Failures: An Empirical Case Study,” First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), 2007, pp. 364–373.

[20] N. Taušan, J. Lehto, P. Kuvaja, J. Markkula, and M. Oivo, “Comparative Influence Evaluation of Middleware Features on Choreography DSL,” The Eighth International Conference on Software Engineering Advances (ICSEA 2013) IARIA, 2013, pp. 184–193.

[21] D. Burdett and N. Kavantzas, “WS choreography model overview,” W3c Work. Draft. W3C, 2004.

[22] R. Dijkman and M. Dumas, “Service-oriented design: A multi-viewpoint approach,” International journal of cooperative information systems, vol. 13, no. 4, 2004, pp. 337–368.

[23] A. Mahfouz, L. Barroca, R. Laney, and B. Nuseibeh, “From organizational requirements to service choreography,” World Conference on Services-I, 2009, pp. 546–553.

[24] I. H. Krüger, “Service specification with MSCs and roles,” IASTED Conference on Software Engineering, 2004, pp. 42–47.

[25] I. H. Krüger, E. C. Nelson, and P. K. Venkatesh, “Service-based software development for automotive applications,” Proceedings of the CONVERGENCE 2004, 2004, pp. 0.

[26] S. Cherrier, Y. M. Ghamri-Doudane, S. Lohier, and G. Roussel, “Services collaboration in wireless sensor and actuator networks: orchestration versus choreography,” IEEE Symposium on Computers and Communications (ISCC 2012), 2012, pp. 411–418.

[27] G. B. Chafle, S. Chandra, V. Mann, and M. G. Nanda, “Decentralized orchestration of composite web services,” Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters, 2004, pp. 134–143.

[28] C. Shekaran, D. Garlan, M. Jackson, N. R. Mead, C. Potts, and H. B. Reubenstein, “The role of software architecture in requirements engineering,” Proceedings of the First International Conference on Requirements Engineering, 1994, pp. 239–245.

[29] R. Ferrari, J. A. Miller, and N. H. Madhavji, “A controlled experiment to assess the impact of system architectures on new system requirements,” Requirements Engineering, vol. 15, no. 2, 2010, pp. 215–233.

[30] H. Cervantes, P. Velasco-Elizondo, and R. Kazman, “A Principled Way to Use Frameworks in Architecture Design,” IEEE Software, vol. 30, no. 2, 2013, pp. 46–53.

[31] R. K. Yin, Case study research: Design and methods, vol. 5. Sage, 2009.

 ISBN: 978-1-61208-367-4