Supporting the Collaborative Development of Requirements and Architecture Documentation

Tom-Michael Hesse
Institute of Computer Science
University of Heidelberg
Im Neuenheimer Feld 326, 69120 Heidelberg, Germany
hesse@informatik.uni-heidelberg.de

Barbara Paech
Institute of Computer Science
University of Heidelberg
Im Neuenheimer Feld 326, 69120 Heidelberg, Germany
paech@informatik.uni-heidelberg.de

***Abstract*—In most software projects, particular requirements significantly drive the design of the software architecture by forcing architectural decisions to be made. As requirements and architecture are refined iteratively, their extensions and improvements need to be aligned continuously. Much research has been conducted to identify such requirements and their impact on architecture. However, it remains a problem how to collaboratively document such requirements and architectural knowledge under development. In particular, knowledge of architectural decisions such as assumptions or alternatives for the system erodes over time and can even vaporize completely. A major reason is the inability to easily manage informality and complexity of knowledge when performing both requirements engineering and architecture design. Therefore, we propose a documentation model for decisions supporting the intertwined documentation of related requirements and architecture knowledge. It provides documentation elements, which are common to both disciplines. In order to support refinement in documentation, knowledge can be iteratively accumulated at different levels of granularity. So the model fits to the twin peaks model of requirements and architecture. In consequence, the comprehension and collaboration between requirements engineers and system architects is improved by negotiating and refining the same documentation together in an ongoing process. We apply our approach to an example in order to demonstrate that it is applicable and useful for managing architectural decision knowledge in relation to the grounding requirements.**

***Index Terms*—Architecturally significant requirements, architectural design decisions, knowledge model, decision documentation**

I. INTRODUCTION

It is widely acknowledged that in software development projects particular requirements drive the systems' architecture. Due to this impact on the software architecture they are called architecturally significant requirements (ASRs) [7]. As the architecture of software can be seen as a set of design decisions [11], architectural design decisions (ADDs) are made to tackle these ASRs. Such ADDs related to ASRs are extremely relevant for the architecture, as they define and explain architectural structures with their origin in particular requirements. We will call all knowledge being or concerning ASRs and ADDs *decision knowledge* in the remainder of this paper.

However, often ASRs are overlooked in the beginning of a project [7] and hints uncovering them are not well documented. Especially in early project stages, requirements and architecture are expressed highly informal [19]. In consequence, also the decision knowledge tends to remain implicit. This informality eventually leads to a loss of decision knowledge, when requirements and architecture evolve. So vague or changing ASRs might be not reflected in the architecture, what results in high costs for later changes [7]. This informality raises the need for a documentation structure, which allows refining given information incrementally. This is the first requirement for our approaches.

Besides informality, another reason hindering the creation, management and use of decision knowledge is the attitude to think of requirements and software architecture as two different and separated kinds of knowledge. As pointed out in [3], such a distinction is arbitrarily and should be avoided. But current approaches either focus on offering models and methods for documenting requirements or for documenting architectural knowledge. Many approaches consider linking both fields, but they do not represent all decision knowledge in a common model for an overall view. So the documentation gap between requirements engineers (REs) and software architects (SAs) is not bridged sufficiently and communication remains being hindered. Hence, a documentation approach is needed, which aims at actively supporting both REs and SAs. Moreover, an explicit and flexible representation for requirements should be given. Those are requirements two and three for the presented approach.

In summary, the first problem is that vague and changing ASRs are not well documented that prevents the architecture from being kept aligned to those requirements. Here, we expect incrementally refined documentation to preserve such knowledge and raise awareness for the alignment. The second problem is that the corresponding ADDs are treated separately from requirements, so that the REs cannot understand them easily and might not be aware of the impact of changing requirements. This is also true for the SAs comprehension of ASRs. So the documentation should address both REs and SAs and provide an explicit representation for requirements, which is applicable by both groups.

The remainder of the paper is structured as follows. In Section 2 we present major parts of the model in detail and give a practice example, in Section 3 we outline the benefits of the model, Section 4 gives a short overview of related work and Section 5 concludes the paper.

II. The Documentation Model

We tackle the problems presented in Section 1 by providing a documentation model for decision knowledge. This model implements the abovementioned requirements and can serve as high-level structure for documenting decision knowledge. Essential parts of the model are depicted in Figure 1. However, the presentation of the full model would exceed the limits of this paper.

In order to provide a better understanding for the model elements, we instantiated them with a practice example. The example decision was taken from [1] and originated from an insurance company. We enriched the example with additional information in order to use all presented model elements. A graphical representation of the example can be found in Figure 2. In the following numbers in brackets refer to an element depicted in this figure.

The example deals with the decision to implement an automated risk approval in an insurance company due to changed internal policies. In addition, it is implicitly stated that the policy change was made because users often had trouble with manual approvals.

This information is elicited and documented by the RE in the first iteration. As a key element, the *DecisionStatement (1)* is employed to represent the decision made with all administrative information needed such as the status of

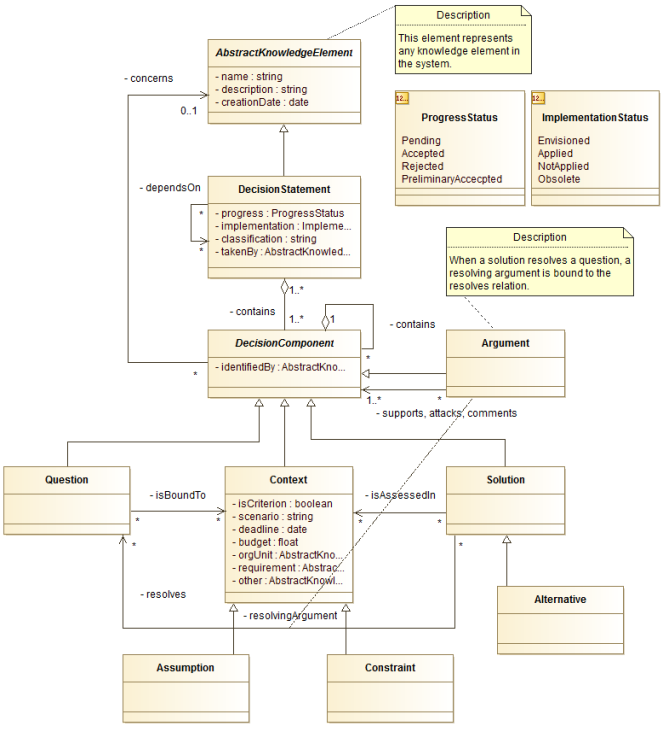

Fig. 2. Typical Elements of the Decision Documentation Model

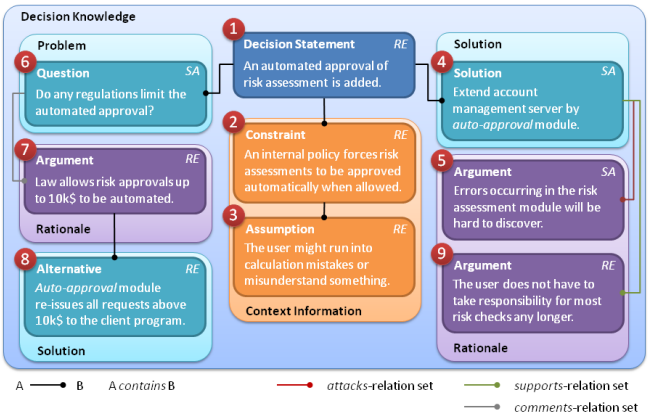

Fig. 1. The Model Applied on Deciding an Automated Risk Approval

decision progress and implementation. It may depend on other *DecisionStatements* (via the *dependsOn*-relation) and can contain any number of *DecisionComponents*, which represent knowledge related to the decision. Each *DecisionComponent* can contain further *DecisionComponents* itself and may be linked to other knowledge elements by the *concerns*-relation, if necessary. *DecisionComponent* is the parent class for all knowledge elements describing the decision more in detail. The children can be grouped in three categories: they may describe the problem or solution of the decision (via *Question* and *Solution*), the decision context and rationales (via *Argument*).

Going back to the example, the RE uses a *Constraint (2)* to describe the additional policy constraint in the environment of the decision. Moreover an *Assumption (3)* is nested in this constraint to explain expected reasons for the constraint and outline its importance. Both elements inherit from the *Context* class which represents knowledge about the environment of decisions and their components. Such context information can drive questions via the *isBoundTo*-relation. Moreover, context knowledge can serve as a criterion for the evaluation of solutions, what is indicated by the *isAssessedBy*-relation.

In the second iteration, the SA makes a solution proposal, reminds of possible negative effects for the system and asks whether law regulations might influence the pending decision. Therefore, a *Solution (4)* is used to express that the account management server of the company is extended by an automated approval module. In addition, an *Argument (5)* is added to this solution with an *attacks*-relation. *Arguments* allow providing explicit rationale for decision knowledge; they can support, attack or comment any part of the decision knowledge. They are also used as an attachment to the *resolves*-relation in order to explain why a particular solution was chosen. The SA employs the *Question (6)* to state his uncertainty towards law restrictions on solutions for an automated risk approval. *Questions* can be applied for any knowledge that might pose a challenge or raise a discussion on the decision.

In the third iteration, the RE has discussed the question with the customer and identified the laws concerned by the solution. So the RE adds an *Argument (7)* commenting the question in order to explain that automated approvals not

exceeding 10k$ are permitted. Moreover, the RE states an *Alternative (8)* to cope with the restrictions on the automated approval. Instead of managing all approvals on the server automatically, approvals above 10k$ are handed over to the client program. Then, the user can make the final decision on the approval. So an *Alternative* is a refinement of solutions showing different solution options and their characteristics. Finally, the RE adds an *Argument (9)* with the *supports*-relation motivating the solution by outlining the advantages for the users.

The requirement in the example turns out to be architecturally significant according to [7]. It is strictly enforced by the internal policy change. It also influences the principal architecture of the system. The risk approval will be realized by adapting the architecture of the account management server. Moreover, a communication channel to the client has to be established, since risks beyond 10k$ need to be approved manually by the user.

The full documentation model contains many additional knowledge elements for a more fine-grained representation of *Question/Solution*-, *Context*- and *Argument*-elements. But the core elements presented in this paper are sufficient to give an impression of the model and to demonstrate its practical use. It should be noted, that none of the elements is dedicated to be exclusively used by REs or SAs. Grouping the elements along their primary use to represent a question, a solution, context or rationale, shall improve the readability and comprehension of the knowledge elements in the figures. But we encourage the usage of all elements by both REs and SAs, whenever the elements fit to the given knowledge. So REs may state solutions like the *Alternative* in (8), as well as SAs are free to raise questions, as shown in (6).

III. Benefits for ASR and ADD Management

In contrast to the current approaches, our documentation model realizes all requirements introduced in Section 1. First, it supports a continuous refinement of knowledge due to the ability to nest *DecisionComponents* iteratively. Second, because there is no border between the problem and the solution description, the documentation model elements are applicable for expressing requirements and architecture. As a result, REs and SAs are supported in continuously negotiating the documentation and collaborating with each other. We will discuss these benefits with respect to the given example.

A. Continuous Refinement of Decision Knowledge

The documentation model allows documenting decision knowledge in an iterative and continuous fashion. The seed for documentation is the *DecisionStatement*, which has a high-level granularity and serves as reminder for the decision by summarizing it. This knowledge could be important for project managers to get a quick overview of the state of requirements and architecture design.

The more elements are added to the documentation, the more fine-grained and specific the knowledge elements will become. An example is the *Solution* and its refinement *Alternative*. Such information can be helpful to SAs being interested in getting to know the latest technical changes. So the hierarchy of knowledge elements represents different levels of granularity.

Through the refinements in the model any information that is recognized as important can be documented, but documentation is not enforced by a static template. None of the knowledge elements presented in Section 2 is mandatory for documentation. So documentation can be focused on the information that matters in a particular context or that is currently present. Hopefully, this decreases the barrier for documenting decision knowledge. In consequence, it is likely that loosing valuable knowledge during the beginning of requirements engineering and architecture design is prevented, because more knowledge is documented explicitly.

Such an iterative documentation approach fits well to the Twin Peaks model, as depicted in Figure 3. Our model contributes to the Twin Peaks model by distinguishing problems and solutions, context and rationales within each of the two peaks. But it does not introduce a border between requirements and architecture, as shown by the refinement iterations of the example. Starting with elements 1 – 3, the RE initially provides decision knowledge about the ASR, which is refined from the SA with his corresponding ADD knowledge in elements 4 – 6. As a reaction of the RE, elements 7 – 9 are documented further refining the decision knowledge.

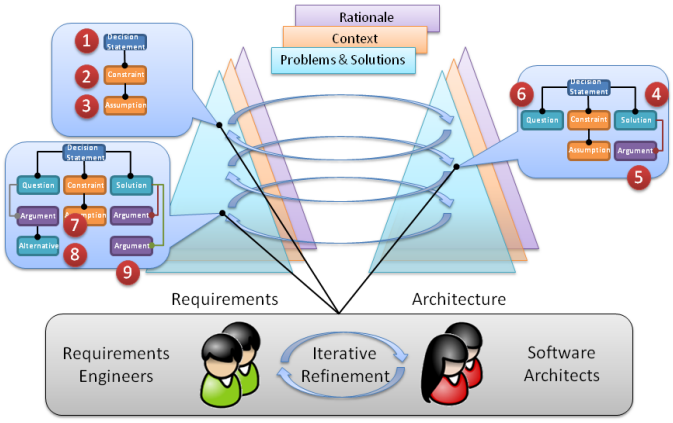

Fig. 3. The Process of Documenting the Automated Risk Approval Mapped to the Twin Peaks Model

In addition, this refinement helps to tackle the issue of an explicit management of assumptions on ASRs, as mentioned in [7]. We present assumptions as first class entities in our model, so that assumptions can attract more attention in documentation activities. In addition, we explicitly provide assumptions in requirements documentation, as they can represent and be related to requirements in our model.

B. Continuous Negotiation and Collaboration Between REs and SAs

Iteratively documenting decision knowledge with our documentation model can help establishing continuous negotiation and collaboration between REs and SAs. Both groups can access and use the same documentation structures. This lowers communication barriers between REs and SAs. Working on the same knowledge will increase the negotiation

processes towards the stored information and their impact on the system. In this way, vague knowledge about requirements is likely to be identified and consolidated faster. We expect this to increase the possibility of having those requirements documented properly that will turn out to be architecturally significant later on.

The negotiation process requires an increased collaboration between REs and SAs. In order to raise awareness for each other [7], the documentation model can serve as a mediator and bring REs and SAs together. On the one hand, the SAs can comprehend the development of requirements, so they can keep their architecture aligned in the resulting ADDs. On the other hand, REs can benefit from questions and solutions introduced by SAs, as they get a better understanding for the impact of the requirements on the system.

IV. Related Work

Our documentation model was inspired by representations for design rationale, such as *Questions, Options, and Criteria* (QOC) by MacLean et al. [14] and the *Decision Representation Language* (DRL) by Lee [13]. The main differences are the explicit and fine-grained modeling of context knowledge and the ability to nest and refine all components of a decision. Moreover, we consider the decision documentation process to be collaborative between different project roles. These aspects also distinguish our model from the specifications in ISO 42010:2011 standard [10]. The standard describes the documentation needs for architectural knowledge, but does not explicitly provide a structure that supports such documentation approaches or collaboration between different stakeholders.

Several approaches exist to support the documentation of ADDs. In Table 1, we present an overview of typical approaches. Many of them were introduced either at the *Quality of Software Architectures* (QoSA) conference or the *Workshop on Sharing and Reusing Architectural Knowledge* (SHARK). For each approach, we investigated to what degree the requirements stated in Section 1 are realized.

Most approaches for documenting architectural design decisions address SAs only. The approach of Burge and Brown RATSpeak with its tool SEURAT focuses on software developers. The approaches of Herold et al. and Mou and Ratiu consider REs explicitly, but lack support for knowledge refinement. Here, our model presents a new approach of documentation, as it takes REs and SAs into account and also provides support for iterative knowledge refinement driven by both roles.

All approaches support the documentation of requirements in different ways. First, RATSpeak/SEURAT, the approach of Tang et al. and Whalen et al. consider requirements implicitly as a part of other knowledge elements, for example in decision rationales, decision motivation or constraints. Second, requirements are represented as attributes of decisions or as links from decisions to requirement artifacts, such as use case description documents. This is done by most approaches, like Archium by Jansen and Bosch, Tyree and Akerman, Kruchten et al. or Capilla et al.. Third, requirements are represented as first-class entity, like in the approach of Falessi et al., PAKME by Babar et al. and Herold et al.; Mou and Ratiu even introduce a formal requirements model in order to automatically transform requirements knowledge into test cases. We decided to implement a hybrid approach by making requirements and their subtypes assumptions and constraints first-class entities with the *Context* element and its subclasses. In addition, external requirements artifacts can be linked to the *Context*. This increases the flexibility in documentation, because all related knowledge can be made available in the model, according to its type. Moreover, the requirements remain explicit and are not hidden in textual descriptions of other elements.

TABLE I. Typical Documentation Approaches for ADDs

Approach	Roles	Representation of Requirements	Knowledge Refinement
RATSpeak [4], SEURAT [5]	Dev.	Implicit: Part of rationale	Nested decisions and alternatives
Archium [11]	SA	Decision attribute: context	Decisions refine other decisions
TyreeAkerman [16]	SA	Decision attribute: Related Requirements	Decisions refine other decisions
Falessi et al. [8]	SA	First-class entity: objective	Static refinement process
Kruchten et al. [12]	SA	Links to requirement documents	Mandatory attributes
PAKME [2]	SA	First-class entity: ASR	Nested rationales
Capilla et al. [6]	SA	Decision attribute: Requirements; Links to requirements documents	Mandatory and optional attributes
Herold et al. [9]	RE, SA	First-class entity: Goal, softgoal, task	None
Tang et al. [16]	SA	Implicit: Part of decision motivation	Element specialization; nested reasons, rationales and outcomes
Zimmermann et al. [20]	SA	Decision attribute: Decision drivers	None
Mou and Ratiu [15]	RE, SA	Formal requirements model	None
Whalen et al. [19]	SA	Implicit: Part of formal architecture constraints	Hierarchy extension

The ability to refine documented knowledge is realized in many different ways, depending on the principles of each approach. The proposals of Herold et al., Zimmermann et al. and Mou and Ratiu do not explicitly consider a refinement process for their documentation. The approaches RATSpeak/SEURAT, PAKME and the approach by Tang et al. employ the nesting of knowledge elements. The approach by Tyree and Akerman and Archium introduce dedicated relations for ADDs in order to express that they refine other decisions. Kruchten et al. and Capilla et al. introduce a set of mandatory attributes for decisions, which have to be set. Optional and user-defined attributes can be added to extend and refine the decision knowledge. Falessi et al. define a static refinement process for their model in order to derive the model elements, but they do not specify how to refine a given element. Tang et al. propose several specializations of high-level model elements, whereas Whalen et al. introduce a hierarchal structure where elements may be added on different

levels. Our approach focuses on the refinement of knowledge through nesting these elements and using specialized model elements. Instead of limiting the nesting to particular elements in our model every *DecisionComponent* may contain other components. So the documentation for all decision elements can be extended iteratively and the documentation process is not restricted by arbitrary model constraints.

V. Conclusion

In this position paper, we have introduced a decision documentation model supporting the collaborative development of documented requirements and architectural knowledge. Decision knowledge can be stored in different knowledge elements that can be nested and specialized as needed. Our model provides two major benefits for the documentation of decision knowledge resulting from ASRs. First, an iterative refinement of knowledge is supported through the nested model elements. In addition, the hierarchy of specialized knowledge elements represents different levels of granularity for the decision knowledge. Second, our model supports the collaboration between REs and SAs, as the model does not distinguish knowledge to belong exclusively to one of both roles. This increases negotiation and collaboration processes between REs and SAs and helps to create an intertwined comprehension of requirements and architecture. Overall, vague and changing requirements knowledge can be identified faster, as both REs and SAs negotiate and refine the same documentation. This leads to a better alignment of the corresponding architectural design decisions.

We are currently working on the implementation of this model in UNICASE [18], which is a knowledge management tool integrating project and system knowledge directly in the Eclipse IDE. We aim at providing tool support for decision knowledge documentation as close to software design and development as possible. We expect our model to be useful for any software project that faces the need for explicit documentation of ASRs and ADDs. However, it should be noted that the model was not yet evaluated within real world case studies. We strive to present such evaluation of our model as soon as we have collected data from practice use through the tool support.

Acknowledgment

This work was partially supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future – Managed Software Evolution.

References

[1] U. Abelein and B. Paech, "A Descriptive Classification for End User - Relevant Decisions of Large-Scale IT Projects" in *6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE'13: ICSE Workshops 2013)*, accepted to appear.

[2] M. A. Babar and I. Gorton, "A Tool for Managing Software Architecture Knowledge" in *Second Workshop on Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent (SHARK/ADI'07: ICSE Workshops 2007)*.

[3] R. C. de Boer and H. van Vliet, "On the similarity between requirements and architecture" in *Journal of Systems and Software*, vol. 82, no. 3, pp. 544–550, 2009.

[4] J. E. Burge and D. C. Brown, "An Integrated Approach for Software Design Checking Using Design Rationale" in *Proceedings of the First International Conference of Design Computing and Cognition*, pp. 557–576, 2004.

[5] J. E. Burge and D. C. Brown, "Software Engineering Using RATionale" in *Journal of Systems and Software*, vol. 81, no. 3, pp. 395–413, 2008.

[6] R. Capilla, F. Nava, and J. C. Duenas, "Modeling and Documenting the Evolution of Architectural Design Decisions" in *Second Workshop on Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent (SHARK/ADI'07: ICSE Workshops 2007)*.

[7] L. Chen, M. Ali Babar, and B. Nuseibeh, "Characterizing Architecturally Significant Requirements" in *IEEE Software*, vol. 30, no. 2, pp. 38–45, 2013.

[8] D. Falessi, G. Cantone, and M. Becker, "Documenting Design Decision Rationale to Improve Individual and Team Design Decision Making: An Experimental Evaluation" in *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering (ISESE '06)*, pp. 134–143, 2006.

[9] S. Herold, A. Metzger, A. Rausch, and H. Stallbaum, "Towards Bridging the Gap between Goal-Oriented Requirements Engineering and Compositional Architecture Development" in *Second Workshop on Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent (SHARK/ADI'07: ICSE Workshops 2007)*.

[10] International Organization for Standardization, International Electrotechnical Commission, "ISO/IEC/IEEE 42010: Systems and software engineering — Architecture description", Genf, CH, 2011.

[11] A. Jansen and J. Bosch, "Software Architecture as a Set of Architectural Design Decisions" in *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pp. 109–120, 2005.

[12] P. Kruchten, P. Lago, and H. Van Vliet, "Building Up and Reasoning About Architectural Knowledge" in *Second International Conference on Quality of Software Architectures (QoSA)*, pp. 43–58, 2006.

[13] J. Lee, "Extending the Potts and Bruns Model for Recording Design Rationale" in *13th International Conference on Software Engineering (ICSE)*, pp. 114–125, 1991.

[14] A. MacLean, R. M. Young, M. E. Victoria and T.P. Moran, "Questions, Options, and Criteria: Elements of Design Space Analysis", in *Human-Computer Interaction*, vol. 6, no. 3–4, pp. 201–250, 1991.

[15] D. Mou and D. Ratiu, "Binding Requirements and Component Architecture by Using Model-Based Test-Driven Development" in *First IEEE International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks)*, pp. 27–30, 2012.

[16] A. Tang, J. Han, and R. Vasa, "Software Architecture Design Reasoning: A Case for Improved Methodology Support" in *IEEE Software*, vol. 26, no. 2, pp. 43–49, 2009.

[17] J. Tyree and A. Akerman, "Architecture Decisions: Demystifying Architecture" in *IEEE Software*, vol. 22, no. 2, pp. 19–27, 2005.

[18] UNICASE, http://www.unicase.org/ (retrieved: June, 2013)

[19] M. W. Whalen, A. Murugesan, and M. P. E. Heimdahl, "Your What is My How: Why Requirements and Architectural Design Should Be Iterative" in *First IEEE International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks)*, pp. 36–40, 2012.

[20] O. Zimmermann, T. Gschwind, J. Küster, F. Leymann, and N. Schuster, "Reusable Architectural Decision Models for Enterprise Application Development" in *Third International Conference on Quality of Software Architectures (QoSA)*, pp. 15–32, 2007.