Architectural Knowledge: Getting to the Core

Remco C. de Boer[1], Rik Farenhorst[1], Patricia Lago[1], Hans van Vliet[1], Viktor Clerc[1], and Anton Jansen[2]

[1] VU University Amsterdam, the Netherlands
{remco, rik, patricia, hans, viktor}@cs.vu.nl
[2] University of Groningen, the Netherlands
anton@cs.rug.nl

Abstract. Different organizations or organizational units are likely to store and maintain different types of information about their software architectures. This inhibits effective management of architectural knowledge. We experimented with a model of architectural knowledge to characterize the use of architectural knowledge in four different organizations. Based on this experimentation we identified four perspectives on architectural knowledge management, and additionally adjusted the model to better align theory with practice. The refined model defines a minimal set of concepts with supposedly complete coverage of the architectural knowledge domain. Because of the minimalistic aspect of the model, we refer to it as a 'core model' of architectural knowledge. Supporting evidence for the validity of our model, i.e. the supposed complete coverage, has been obtained by an attempt to falsify this claim through a comparison with selected literature. Application of the core model to characterize the use of architectural knowledge indicates possible areas of improvement for architectural knowledge management in the four organizations.

1 Introduction

The notion of software architecture is one of the key technical advances in the field of software engineering over the last decades. The advantages of using an explicit software architecture include early interaction with stakeholders, its basis for a work breakdown structure, and the early assessment of quality attributes [1]. Although considerable progress has been made in this area, we still lack techniques for capturing, representing and maintaining knowledge about software architectures.

Various authors (e.g. [2,3,4]) address the notion of 'architectural knowledge' and provide a model of what this notion entails. Key elements in all models are design decisions and their rationale. However, different authors use different words for what might be the same. For example, some models consider design decisions, others architectural decisions, but it is hard to determine whether these actually denote the same concept.

Having different notions of what architectural knowledge entails can hamper effective management of that knowledge. If, for instance, different organizations – or even departments within a single organization – use different concepts to communicate architectural knowledge, terminological misunderstandings may arise. Sharing architectural knowledge between these parties then becomes very hard, if not impossible. We need a

S. Overhage et al. (Eds.): QoSA 2007, LNCS 4880, pp. 197–214, 2007.
© Springer-Verlag Berlin Heidelberg 2007

model of architectural knowledge that acts as a common frame of reference and enables architectural knowledge sharing.

The question we address in this paper is what this model should entail. As an answer, we propose a model of architectural knowledge that has maximal expressivity in the architectural knowledge domain and functions as a reference model for sharing architectural knowledge. Real-life models of what architectural knowledge entails can be expressed in the form of extensions to this model. These extensions are domain-specific, organization-specific, or both.

2 Related Work

In an overview of the maturation of the software architecture field [5], Shaw and Clements conclude with an outlook on future work in software architecture research. Promising topics mentioned include a focus on architectural design decisions and their link to quality attributes, and the organization of architectural knowledge to create reference materials. Our work serves both these goals.

The research field already shows increasing focus on the management of architectural knowledge (i.e. knowledge pertaining to a particular software architecture), such as architectural design decisions and their rationale [3,6,7,8]. A growing number of researchers acknowledges that a software architecture can – or should – be viewed as the collection of architectural design decisions [9], or as the design decisions plus the resulting design [4]. Others target tracing architectural decisions to concerns [10], or link business goals to the software architecture [1].

Hofmeister et al. define architecting as an iterative process in which the architecture 'grows' over time as architects perform architectural activities, such as analysis, synthesis, and evaluation [11]. In our research we build further on this view, by considering the iterative nature of architecting as a 'decision loop'. We focus not only on the design decisions themselves, but also on the result of this iterative process – the architectural design – which is reflected in various design artifacts such as architectural descriptions.

In recent years, several other models or frameworks have been proposed to capture architectural knowledge. Akerman and Tyree propose an ontology that focuses on architectural assets, architectural decisions, stakeholder concerns and an architecture implementation road map [12]. A framework for capturing and using architecture design knowledge is proposed by Ali Babar et al. in [2].

All methods and frameworks described above share a common understanding of what 'architectural knowledge' is, or should entail. Then again, each of the methods and frameworks has a different focal point and may use terminology differently from others. In an attempt to aggregate the common understanding, while allowing for different specializations of the central concepts, we have constructed a reference model of architectural knowledge that is described in Section 5.

3 Research Methodology and Structure of This Paper

The structure of this paper, schematically depicted in Fig. 1, is tightly coupled to the research approach we followed. In the remainder of this section we outline the research

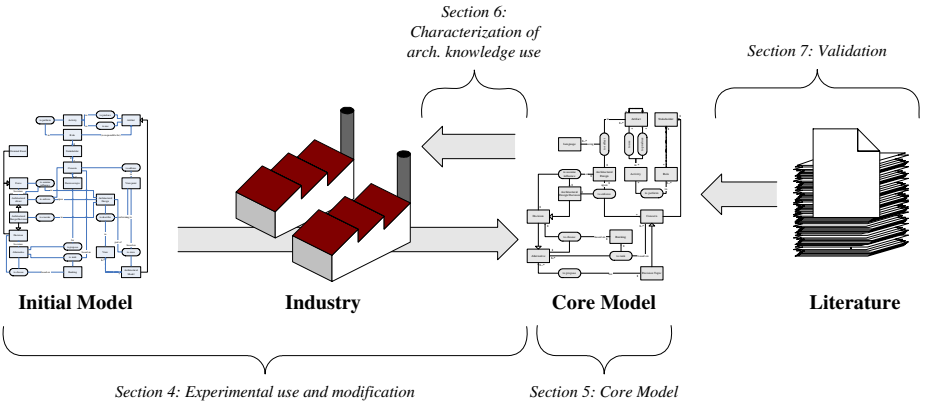

Fig. 1. Structure of this Paper

methodology we employed, the steps we followed, and in which sections the respective results are elaborated upon.

The research methodology that we followed can best be described as an instantiation of action research. Action research is an iterative research approach in which the researcher actively participates in the studies he performs. The researcher wants 'to try out a theory with practitioners in real situations, gain feedback from this experience, modify the theory as a result of this feedback, and try again' [13].

Our research commenced with a 'theory' of architectural knowledge that stemmed from our earlier work [14]. This initial model of architectural knowledge was an abstract conceptualization of the architectural knowledge domain. We experimented with the model, and tried to characterize the use of architectural knowledge by (and together with) four industrial partners. Experience from those characterization attempts taught us that there were a number of mismatches between our theory and industrial practice. Reflection on the apparent mismatches led us to conclude that our model should exhibit a number of properties in order to overcome those mismatches. This reflection process is elaborated further in Section 4. In order to accommodate for the desired properties identified, we refined the initial model and arrived at a new 'version' of our theory of architectural knowledge: a core model of architectural knowledge presented in Section 5.

With the mismatches between theory and practice removed, we could successfully employ our core model of architectural knowledge to characterize the use of architectural knowledge by the four partners. This characterization, which is the subject of Section 6, led to a number of hypotheses regarding the probable cause of problems with architectural knowledge management in the collaborating organizations. We plan to alleviate the identified problems by removing the probable causes in the near future. This illustrates the iterative nature of action research, where the result of the action research cycle we performed is input for the next cycle.

Since we want our model to be useful as a reference model to align different architectural vocabularies, we believe our model can be regarded as 'valid' when concepts from different architectural approaches can be expressed using terms from the model.

Unfortunately, it is impossible to prove that our model is valid in this sense. However, we can make the validity of our model plausible by trying to falsify our model by comparing the model with (accepted) literature. This falsification attempt, which falls outside the scope of the action research cycle itself, is discussed in Section 7.

4 A Theory of Architectural Knowledge: Experimental Use and Modification

The goal of our research was to characterize the use of architectural knowledge in four different organizations. The four organizations that participated in our research can be described as follows:

- **RFA** is a large software development organization, responsible for development and maintenance of systems for among others the public sector.
- **VCL** is a large, multi-site consumer electronics organization where embedded software is developed in a distributed setting.
- **RDB** performs independent software product audits for third parties.
- **PAV** is a large scientific organization that has to deal with software development projects spanning a long time frame (up to a period of more than ten years).

The theory that we used to characterize the organizations' architectural knowledge use was a model of architectural knowledge that is presented in detail in [14]. This model had been constructed to structure software architectural knowledge in such a way that it is clear what can exist and what can happen during the architecting phase of a software development project. We aimed to use this model as an abstract view of the architectural knowledge domain, to allow for clean reasoning about the use of architectural knowledge in the four organizations.

Experience with the model from [14] in the four industrial organizations taught us that the model did not entirely fit all organizations. The original model highly conformed to the IEEE-1471 standard for architectural description [15]. IEEE-1471 prescribes the use of so-called 'Viewpoints' to describe the architecture from the perspective of different stakeholders. The resulting 'Views' (partial descriptions of the architecture) are aggregated in a single architecture description. Although stakeholders and their concerns play a key role in any software architecting process, the tight coupling of the model to IEEE-1471's Views and Viewpoints turned out to be a mismatch with most organizations' practice. In hindsight this need not come as a big surprise, since organizations can (and do) use other approaches for documenting their architectures, which need not coincide with the IEEE-1471 way.

Although the model from [14] did not entirely fit all organizations, diagnosis of the use of architectural knowledge in those organizations at least showed that each of the organizations has its own perspective on architectural knowledge management, resulting in different issues at each of the organizations. The central issue within RFA was how to *share* architectural knowledge between stakeholders of a project. The main question within VCL was how *compliance* to architectural rules can be enforced in this multi-site environment. RDB was mainly concerned with how auditors can *discover* the architectural knowledge they need to do a proper audit. The main challenge for PAV was how to

improve *traceability* of its architectural knowledge. While the mismatches between theory and practice still prevented us from pinpointing the exact areas of improvement, at least we had an idea where to search for those areas in a next research iteration. However, this required that we removed the identified mismatches to further align theory with practice.

From a closer inspection of the mismatching concepts we learned that those concepts could either be expressed in terms of other concepts already present in the model, or as more generic concepts that *are* used by the organizations. This led us to believe that we should strive to construct a model of architectural knowledge that is both minimalistic and complete. We believe the model can be regarded as 'complete' if there are no concepts from other approaches that have no counterpart in the model. If there turns out to be such a missing concept, our model should be extended. With 'minimalistic' we signify the feature that it should not be possible to express some concepts from the model in any other concepts from the model.

Based on these insights we modified the initial model from [14] to obtain such a model that is both complete and minimalistic. Especially because of this latter feature, we refer to our model as a core model of architectural knowledge; elements that can be modeled in terms of core elements do not belong to the core.

5 A Core Model of Architectural Knowledge

Our core model of architectural knowledge is depicted in Figure 2. As a result of the minimalistic aspect of this model introduced in Section 4, the core model leaves room for the use of different architecture description methods, including IEEE-1471. This contrasts with the model from [14] in which the use of IEEE-1471 was assumed and which therefore did not match those organizations that use other architecture description methods.

In our core model of architectural knowledge, the concepts of Stakeholder and Concern coincide with the, widely accepted, definitions of these terms in IEEE-1471: a stakeholder is "an individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system" [15]. Both IEEE-1471 concepts of *Architectural Model* and *View* are subsumed in our notion of Artifact, i.e. an inanimate information bearer such as a document, source code, or a chapter in a book. Storing or describing the Architectural Design in either of these artifacts can be abstracted to a single action 'to reflect'. The Architectural Design can be reflected using different Languages, including models, figures, programming languages, and plain English.

Constructing an architectural design is essentially a decision making process. In our core model, decision making is viewed as proposing and ranking Alternatives, and selecting the alternative that has the highest rank, i.e. the alternative that, after careful consideration based on multiple criteria (i.e. Concerns), is deemed to be the best option available with respect to the other alternatives proposed. It is especially this process of proposing, ranking, and selecting which is hard to articulate and distinguishes the good architects from the weaker. The chosen alternative becomes the Decision. The alternatives that are proposed must address the Decision Topic, and can be ranked according to how well they satisfy this and other concerns. We view Decision Topic as a special type

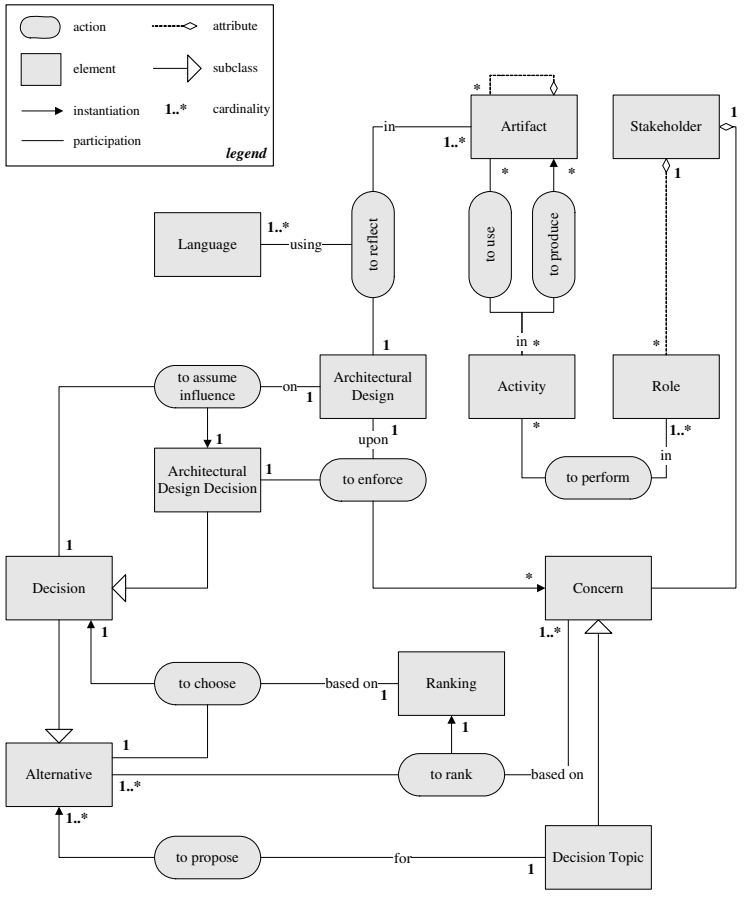

Fig. 2. Core Model of Architectural Knowledge

of Concern, namely a Concern for which a Decision must be taken. Example concerns for which no further decisions need to be taken – and which hence are no decision topics but do need to be taken into account when evaluating (ranking) proposed alternatives – are constraints, such as budget constraints, technological limitations, et cetera.

Architectural Design Decisions are defined as those Decisions that are assumed to influence the Architectural Design and can be enforced upon this Architectural Design, possibly leading to new Concerns that result in a need for taking subsequent decisions. This 'decision loop' captures the relations between subsequent Architectural Design Decisions. This loop also corresponds to the 'divide and conquer' technique of decision making, in which broadly scoped decisions are taken which may result in finer grained concerns related to the broader concern. Furthermore, it enables in theory traceability from concerns through decisions to artifacts, although this very much depends on whether those traces have been captured in the reflection of the architectural design. Note that architectural design decisions need not necessarily be 'invented' by the

architect himself; architectural patterns, styles, and tactics are examples of architectural design decisions (or, more precisely, alternatives) that are readily available from other sources. The 'decision loop' described above also captures the rationale of an architectural design; the answer to the question why an architectural design is the way it is. Rationale is in our opinion extremely interwoven with all elements in this loop, and is therefore not represented as a distinct element in our model.

The Architectural Design (often called 'architecture' for short) is the result of all architectural design decisions. Note that reflection of (part of the) Architectural Design is not limited to a single Artifact: a single Architectural Design Decision might for instance be represented in the architecture description as well as impact the source code. Artifacts themselves can again be composed of various (sub)Artifacts, e.g. chapters in a document, or methods in a class. The concepts Role and Activity are borrowed from SPEM, which defines the software development process as "a collaboration between abstract active entities called process roles that perform operations called activities on concrete, tangible entities called work products" [16]. The 'work product' from SPEM resembles our notion of Artifact. The latter is in our opinion a better known and widely accepted concept in Software Engineering.

6 Core Model Application: Characterization of Architectural Knowledge Use in Four Industrial Settings

We initially started our research with the goal to characterize the use of architectural knowledge in four industrial organizations. Although we were able to discover four different perspectives on architectural knowledge within those organizations (see Section 4), the mismatches of our initial model of architectural knowledge with the observed practice in those organizations hampered a further diagnosis of the problems those organizations encounter. Since those mismatches have been removed in our core model of architectural knowledge, it is interesting to see how the organization-specific 'models' of architectural knowledge of all four organizations can be expressed in terms of the core model. From a superficial look, each of the organizations appears to use architectural knowledge very differently. Alignment of the organization-specific models to the core model, however, allows for a more fundamental characterization of how the organizations perceive and use architectural knowledge.

The use of architectural knowledge in RFA and VCL is mainly located in the upper 'description' part of Figure 2, i.e. the reflection of architectural design decisions in artifacts. The use of architectural knowledge within RDB and PAV is positioned more in the lower 'decision' part, i.e. the decision making process reflected in the decision loop. We hypothesize that the problems that the organizations experience in managing architectural knowledge are partially due to their focus on only a part of the theory of architectural knowledge as expressed by our core model.

An overview of the result of the four characterizations is provided in Table 1. In this table we list for each of the organizations their prevalent perspective on the use of architectural knowledge, the main architectural knowledge concepts encountered (both organization-specific and at a core level), the hypothesized cause for the diagnosed

Table 1. Diagnosis of Industrial Problems with Architectural Knowledge Management

	RFA	VCL	RDB	PAV
Perspective	Sharing	Compliance	Discovery	Traceability
Main org. concepts	Design choices, Principles, Starting Points, Prerequisites	Architectural rules	Quality criteria, Quality in use	Knowledge entities
Mainly used core concepts	Arch. design decisions	Arch. design decisions, *to enforce*	Arch. design decision, Concern, Arch. design, *to reflect*	Concern, Decision topic Alternative, Arch. design decision
Problem	Ambiguous terminology	No sense of urgency regarding compliance with architectural rules	Implicit relation between architecture and "quality"	Lack of traceability between knowledge entities
Hypothesized cause	Decision making process not captured	Tacit decision making process	'Quality' and related arch. knowledge not confined to a single artifact	Implicit relations between artifacts
Possible solution	Explicit focus on relations between decisions through iterative "decision loop"	Explicit focus on rationale of decisions through 'decision loop' elements	Uncover architectural design decisions, their cause, and their effect on the software product	Annotate architecture documents with specific knowledge entities

problems, and a possible solution to this problem. In the following subsections, we further elaborate on these aspects for each organization in turn.

6.1 RFA: Development Organization

RFA is a large development organization that develops and maintains software systems for among others the public sector. These systems are typically critical for the public, large in size and complexity, and long lasting. Because of the size of the organization and the projects, this organization focuses on how to effectively share architectural knowledge. To this end, RFA developed its own methodology and tooling to aid the software architects in creating and documenting architectural knowledge by means of architectural descriptions.

Architectural descriptions within RFA basically consist of a number of views based on predefined *viewpoints*, and a set of specific *architectural models* such as an object model, a functional data-model, etc. These models reflect a number of *architectural choices*, which are based on *decisions* that relate to *business objectives*. The choices take into account the *design principles*, *starting points*, and *prerequisites* to which the architect needs to adhere to when designing the software architecture, as well as the *stakeholder concerns*.

An example of a business objective documented in one particular architectural description is: *"The data of the subsystems needs to be easily accessible"*. A stated design principle based on this objective is *"The system needs to be accessible using web services as well as the Enterprise Service Bus"* and the final architectural choice made based on this principle is *"The system information exchange uses InfoMessaging and MS.NET Web Services"*.

We interviewed architects and managers from RFA, who in these interviews acknowledged that they are currently struggling with the different concepts, their relations and their more effective usage. This impairs effective sharing of architectural knowledge, since architects are unsure which concepts to use to describe their architectural

design. As a result, readers are unsure where in the architecture description they can find the information they are looking for.

If we express the organization-specific terminology in terms of core concepts, an interesting pattern emerges. The 'different' notions of business objectives, design principles, and architectural choices all are in fact Architectural Design Decisions, which are somehow related to each other. However, the organization's methodology does not define very concrete guidelines to distinguish between those decisions. RFA's struggle with terminology might partially be blamed on the use of different terms for the same architectural knowledge concept without a good definition of the discriminative features for these terms.

The explicit use of architectural knowledge within this organization can be primarily found in the 'description' area of the core model: reflecting architectural decisions in an architectural description. The decision making process – reflected in the decision loop in the model – is left implicit by the organization's methodology.

Our core model captures relations between different architectural design decisions in the decision loop, where a certain architectural design decisions leads to a new concerns that in turn leads to new decisions. A more explicit focus on this loop would help defeat the ambiguity in terminology within RFA. A design principle could then for instance be defined as a decision taken because of new concerns introduced by a business objective. In the example above, the concern introduced by the business objective would be the need for accessibility. Architectural choices are related to design principles analogously.

6.2 VCL: Consumer Electronics

VCL is a large organization within the consumer electronics domain. This organization has arranged software development along subsystems. A release of the software for a consumer electronic product consists of integrating the relevant subsystems. Each subsystem is developed by a small, dedicated development team. The teams are located at multiple, geographically spread development sites.

This arrangement of the software and the software development activities demands guidelines to maintain the subsystem-based software architecture. To this end, a central architecture team issues architectural rules: a set of principles and statements on the software architecture that must be complied with throughout the organization.

Architectural rules originate from various *issues* that influence VCL's software development, such as defects identified in subsystem releases, change requests, or additional requirements. These issues need to be addressed in the software architecture. *Solutions* to these issues – which affect all subsystems – are captured in text-based documents. These documents are sent to all development teams as *architectural rules* that need to be adhered to.

The creation of architectural rules can be expressed in terms of the core model. The issues identified by the various stakeholders correspond to Concerns of Stakeholders. The architectural rules, i.e. the chosen solutions for these issues, are Architectural Design Decisions that are enforced upon the Architectural Design through dissemination of the Artifacts in which they are reflected.

Although VCL is similar to RFA in that the focus of architectural knowledge use is on the 'description' area of the core model, development in teams at distinct locations

should put a particular emphasis on *enforcement* of architectural design decisions. However, once the architectural rules have been disseminated to the individual development teams, adherence to the rules on the subsystem architectures is the responsibility of the teams themselves. In practice, some of the rules are disregarded by the teams. Our core model suggests that one of the reasons for this might be the fact that only the architectural design decisions themselves are being communicated, while the decision making process itself remains tacit. This lack of insight into the reason of the architectural design decisions taken make that the development teams do not feel a sense of urgency regarding compliance with these decisions. We believe that more information about the rationale of the architectural rules, including the concerns that led up to the decisions, increases this sense of urgency with the developers.

6.3 RDB: Quality Audits

RDB is a company that performs independent software product audits for third parties. A software product audit consists of comparison of *quality criteria* with the actual software product. Most quality criteria assess the effects of *architectural decisions* as reflected in the *software product artifacts*. A quality criterion might for instance be *"All access to data in a relational database should take place through dedicated data access objects. No direct communication of business objects with the database is allowed"*.

The customer that acquires a software product expects this product to have a certain 'quality in use' [17]. Given the concern 'quality in use', there are various quality characteristics for which quality criteria must be selected. For instance, the criterion that all data access must take place through data access objects favors the maintainability of the software product over its efficiency. Selection of quality criteria therefore depends on the relative importance of each of the quality characteristics indicated by the customer. The example criterion will only be selected if maintainability of the software product is indeed more important to the customer than efficiency.

The problem an auditor faces when performing a software product audit is that architectural design decisions and the resulting architectural design are usually not reflected in a single artifact. Even if there is a document called 'the architecture description', architectural decisions impact other product artifacts (e.g. documentation and source code) as well. There is no guarantee that the information in an architecture description is complete, or even up-to-date.

The reflection of the effects of architectural decisions in different software product artifacts can be readily identified in our core model. The Language used to reflect Architectural Design in Artifacts can be a natural language (e.g. English in software product documentation), but also a programming language (source code) or graphics (e.g. diagrams and figures in a software architecture description). A more interesting and less apparent mapping is the mapping of 'quality' to the core model.

In terms of our core model, *quality in use* is a Concern of the customer, who is a Stakeholder. The quality *characteristics* and *subcharacteristics* are Decision Topics for which *quality criteria* are proposed and selected. The proposed criteria are the Alternatives. Selection of quality criteria is based on their impact on the quality in use – the Concern– in terms of prioritized (sub)characteristics, as indicated above. In this way, trade-off analyses are being made regarding conflicting criteria. The chosen quality

criteria describe the architecture in terms of how it ought to be. In other words, quality criteria are a special type of Architectural Design Decisions: decisions that are expected to have influenced (rather than enforced upon) the Architectural Design.

In RDB the relation between architecture and quality is not obvious at first sight. The core model helps us to describe quality in terms of architectural decisions and their effect on the software product, and shows us that quality criteria that apply to the architectural design are themselves architectural decisions. Using the core model, we can express a software product audit as a comparison of two types of architectural knowledge: architectural knowledge that is present in the software product, reflected in the Artifacts that make up the product, and architectural knowledge that is expected by the customer, reflected in quality criteria. This makes it more apparent which architectural knowledge is most important for a software product audit: the architectural design decisions as well as their cause (e.g. stakeholder concerns and trade-offs) and effect (e.g. constraints on subsequent decisions) should be discovered from multiple artifacts for an effective assessment of a software product's quality.

6.4 PAV: Scientific Research

PAV is a scientific organization that is involved in the development of large software-intensive systems, used for scientific research. One of their projects is the development of a highly distributed system that collects scientific data from around 15.000 sources, distributed over 77 different stations, each source generating around 2 Gbps of raw data. The challenge for this system is to communicate and process the resulting 30Tbps data stream in real-time for interested scientists.

In this project, architectural decisions need to be shared and used over a time span of more than 25 years. This is due to the long development time (more than 10 years), and a required operational lifetime of at least 15 years. The organization is judged by external reviewers on the quality of the architecture and the outcome of these reviews influences the funding, and consequently the continuation, of their projects. Therefore, it is of paramount importance to keep the system architecture at a high quality. In order to achieve this purpose, PAV needs to evaluate at all times the design maturity, the completeness, the correctness and the consistency of the architecture.

The evaluation of the architecture is to be performed at the level of *knowledge entities*, which are units of architectural knowledge shared and communicated among the project *stakeholders*. There are four different types of *knowledge entities*: *problems* that state how specific functional requirements or quality attributes must be satisfied; *concerns* that comprise any interest to the systems development, its operation or any other aspect that is critical or otherwise important to one or more stakeholders; *alternatives* that solve the described problem, potentially in different ways and with different consequences; *decisions* that denote the selection of one among multiple alternatives.

During the architecting process, the architect takes a number of architectural decisions that are gradually being refined into more low-level, technical decisions. The lowest level of an architectural decision is called a *specification*, and the architecting process finishes when all architectural decisions have been refined into specifications.

Knowledge entities can be expressed in *artifacts* that are documents in electronic or printed format. The organization also considers *artifacts* of smaller granularity, called

artifact fragments, such as individual sections, paragraphs or pictures in a document, in order to be able to trace fine-grained *knowledge entities* within a single document. Finally, it is of high importance for the organization to keep trace of how the *requirements*, described in the *requirements specification* document are satisfied in the *software architecture* document. Some *requirements* have associated *risks*.

We can express PAV's organization-specific terminology in terms of our core model. A *problem* is being solved by the alternatives, which coincides with the core model concept of a Decision Topic. The *knowledge entity* on the other hand is a generalization of four core model concepts, namely Concern, Alternative, Decision Topic and Decision. A *specification* is a special, refined case of a Decision. An *artifact fragment* is an Artifact contained in other Artifacts. *Requirements* and *risks* are special types of Concerns that need to be taken care of in the decision making process.

During our research in PAV we found that – although many of the core concepts are present – most relations between artifacts remain implicit, potentially leading to traceability issues. The mapping between the core model and the concepts specific to this organization brings to the architect's attention that four of the fundamental core model concepts, namely Concerns, Decision Topics, Alternatives and Decisions are in fact special cases of *knowledge entities*. With this in mind architects can annotate architecture documents with higher level of detail by taking into account the different types of knowledge entities. A more explicit focus on the core model's 'decision loop', and in particular the individual elements that make up this loop, is likely to result in better traceability.

7 Core Model Validation: Attempted Falsification through Literature

The industrial experiences described in Section 6 showed a practical application of our core model of architectural knowledge. In this section we determine the core model's theoretical significance by comparing its concepts to architectural knowledge concepts used in accepted software architecture literature.

As defined in Section 4 our core model is *minimal* in the sense that it is not possible to express some concepts in any other concepts, and *complete* in the sense that there are no concepts from other approaches that have no counterpart in the model. Unfortunately, it is impossible to 'prove' that our model exhibits the desired features of being complete and minimal. The best we can do is to search for counterexamples that prove our model does *not* exhibit those features, thereby demonstrating that our model is *not* valid. If we don't succeed in this falsification attempt, we accept that as supporting evidence for the validity of our model.

To properly apply the falsification approach on our core model, we have mapped on our model the complete set of concepts from three different terminological frameworks for architectural knowledge well-known from literature. Each of these frameworks has a slightly different perspective on architectural knowledge: IEEE-1471 [15] targets architectural descriptions, Kruchten's ontology [18] focuses on architectural design decisions per se, while Tyree and Akerman [6] provide a template to capture architectural

Table 2. Falsification Attempts on Core Model using Software Architecture Literature

Core concept	IEEE-1471 [15]	Kruchten's ontology [18]	Tyree's Decision Template [6]
Stakeholder	*Stakeholder*		
Concern	*Concern, Environment, Mission*	*Requirement, Defect, Risk, Plan*	*Assumption, Constraint, Requirement*
Decision Topic		*scope*	*Issue, Group*
Alternative		*Idea, Tentative*	*Position*
Ranking			*Argument*
Arch. Design Dec.		*Decision*	*Assumption, Decision, Principle*
"decision loop"	*Rationale*	*relationships*	*Related decisions / requirements / principles*
Arch. Design	*Architecture*		
Language	*(Library) viewpoint*		
to reflect	*(Library) viewpoint*	*trace from/to*	*related artifacts*
Artifact	*System, View, Model, Arch. description*	*Technical artifact*	*Artifact*

design decisions – thereby relating architectural decisions to architectural descriptions. Together, these perspectives cover all 'corners' of our core model.

Our falsification attempts are summarized in Table 2, which shows the mapping between core model concepts and the concepts of the three terminological frameworks for architectural knowledge. We failed to find any concepts that do not fit our core model; we accept this result as support to the claim of validity of our core model. In the following subsections the relationships between core model concepts and concepts of each of the literature frameworks are elaborated upon.

7.1 Architectural Descriptions: IEEE-1471

IEEE-1471 prescribes the use of so-called *view*s to describe an architecture. Views are reflections of part of the architectural design according to a particular perspective, or *viewpoint*. A viewpoint defines "the language, modeling techniques, or analytical methods to be used in constructing a view based upon the viewpoint" [15]. In other words, a viewpoint defines the Languages to use as well as how to *reflect* the architectural design in a view. The IEEE-1471 terms *model* and *view* are both subsumed in the core model concept Artifact.

A *library viewpoint* is a *viewpoint* that is defined elsewhere, i.e. a specialized instance of a normal viewpoint. The core model captures the *rationale* of an architectural design decision in the trajectory from Concern and Decision Topic through ranking of Alternatives to the eventual choice of the Decision.

In IEEE-1471 terms, a *system* has an *architecture*, reflected in the core model as an Architectural Design that is reflected in a set of Artifacts, which together correspond to the IEEE *system*. The *architectural description* is a particular Artifact that conforms to the IEEE prescription of how the Architectural Design should be reflected, i.e. using a *viewpoint* as Language. The *environment* in which a system operates determines the setting and circumstances of developmental, operational, political, and other influences upon that system. These influences are represented by Concerns in the core, as described in Section 5.

Finally, a *mission* is defined as 'a use or operation for which a system is intended by one or more stakeholders to meet some set of objectives'. This is a special case of a Concern, i.e. 'an interest which pertains to the system's development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders'.

7.2 Ontology of Architectural Design Decisions

In [18], Kruchten defines an ontology of architectural design decisions. Kruchten argues that 'design decisions deserve to be first class entities in the process of developing complex software-intensive systems' and proposes a model to do so. Due to space restrictions, this section only highlights some of the concepts from the ontology to capture the spirit of this model and its relation to the core model. For a more elaborate discussion, please refer to [19].

Kruchten defines a number of attributes of architectural design decisions, all of which can be mapped to concepts in our core model. The *scope* of a decision, for example, can be mapped on the Decision Topic concept. In our core model, Concerns consist of one or more Decision Topics for which a Decision must be taken. These Decision Topics are concrete subjects for which a solution is proposed. A single Decision Topic limits the scope of a Decision to the concrete subject it represents.

The evolution of design decisions is captured in the *state* attribute. In the early decision making phase, for instance, *Idea*s and *Tentative* decisions (i.e. decisions with a state 'idea' or 'tentative') correspond to the core concept of an Alternative. Both are not yet decisions, and might never become one. Tentative decisions can be used in running 'what if' scenarios, i.e. a ranking of different ideas.

The ontology further defines a number of relations between decisions. All these relationships are manifested in the 'decision loop' in the core model, described in Section 5. For example, the relation 'decision A *constrains* decision B' implies that B is tied to A and must be in the same state as A. For instance, 'Must use J2EE' constrains 'use JBoss as Application Server'. In terms of the core model, the Decision 'Must use J2EE' introduces a new Decision Topic 'which application server to use?'. The Alternative 'JBoss' could not have been chosen without the decision to use J2EE.

Besides relations between decisions, Kruchten names several relations with 'external artifacts'. Design decisions *trace from* technical artifacts upstream: requirements and defects (i.e. Concerns in the core model), and *trace to* technical artifacts downstream: design and implementation artifacts (i.e. Artifacts in the core model). They are also traceable to management artifacts, such as risks and plans (again Concerns). Finally, Kruchten notes that it may be useful to track which portions of the system are *not compliant with* some design decisions. In the core model, this non-compliance corresponds to a reflection in Artifacts of an Architectural Design upon which some Architectural Design Decisions have not yet been enforced.

7.3 Documenting Design Decisions

In [6], Tyree and Akerman consider important decisions as the major forces that drive architecture. They present a template that can be used to document design decisions. According to this template, *assumptions* and *constraints* limit the alternatives that can

be selected. Assumptions are Decisions that are assumed to have been taken and to influence the Architectural Design, often resulting in new Concerns for Stakeholders. Constraints are posed by decisions already taken, reflected in new Concerns. These new Concerns are to be taken into account when ranking Alternatives for a Decision Topic.

During the architecting process, an architect comes up with *positions* for a certain *issue*. Ultimately, one of the *positions* is chosen based on some *argument*. This position becomes the *decision*. This sequence of steps is also visible in our core model, as the proposal of a set of Alternatives, out of which one Alternative is chosen as Decision.

A *group* can be used to organize a set of decisions based on their topic (e.g. integration, presentation, etc.). In our core model, Decision Topics (i.e. concrete subjects for which a solution is proposed) correspond to the *group* concept from [6]. The example template in [6] lists the positions (i.e. Alternatives) 'Rearchitect existing batch logic in System A', 'Extend System B to handle a new product type', and 'Develop a replacement for System A' for the issue 'Current IT infrastructure doesn't support interactive approval functionality for most financial products', with a grouping labeled 'System structuring'. The example template clearly shows the proposed *positions* all target the group (i.e. Decision Topic) 'System structuring'.

According to [6], a *decision* states the architecture's direction. This corresponds to our notion of an Architectural Design Decision that is enforced upon the Architectural Design. The template has room for the documentation of *related decisions*, *related requirements*, *related artifacts*, and *related principles*. The way in which Decisions (that include principles), Artifacts, and Concerns (i.e. requirements) can be related has been extensively described in Section 7.2.

8 A Vision on Architectural Knowledge Sharing

We can look at the core model from two perspectives, namely data integration and service integration. For data integration the core model becomes a reference model for sharing architectural knowledge. For service integration, it provides the means to integrate the services that a grid infrastructure may provide.

Having a core model of architectural knowledge has a number of advantages. First of all, from a data integration perspective, the core model defines a vocabulary for architectural knowledge: the minimal set of common notions that is needed when architectural knowledge has to be made explicit. Terminology, processes, and concerns particular to a specific organization or domain can be expressed in terms of core model concepts. Metaphorically speaking, the organization-specific terminology lies like a shell around the core model. One can even envision multiple layers of shells, for instance when an organization defines its own methodology (the outer shell) as an extension to the IEEE-1471 standard (the inner shell). Thanks to this separation between core model and shells, we gain in terminological stability (the core model acts as a reference model used by all companies), extensibility (the architectural knowledge of new companies or domains can always be added as a new shell without any increase in complexity), and reuse (by adding new shells, the architectural knowledge vocabulary is incrementally enriched).

Moreover, with a shared core model it becomes easier to agree on a common terminology. This common terminology sticks to the essence and neglects the specific

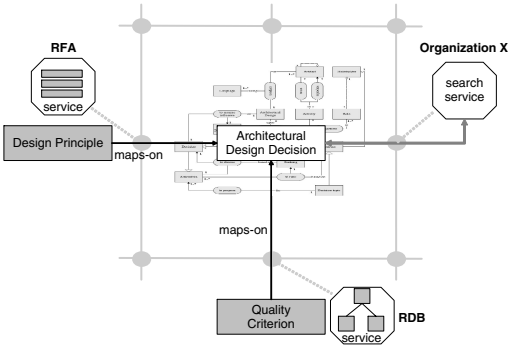

Fig. 3. Sharing Architectural Knowledge in a grid-setting

concepts delegated to the shells. These benefits are reaped whenever multiple departments in the same organization, or even different partner organizations, have to collaborate in the same software project: terminological misunderstandings are avoided.

Furthermore, from a service integration perspective the core model can be the means to integrate the services that a grid infrastructure may provide. These services may 'speak the same language' by exchanging data expressed in concepts from the core model. A direct benefit of this language uniformity is that the core model, being shared among multiple sites, realizes a more generic architectural style aimed at integration via an enterprise data model [20]: the enterprise data model (i.e. our core model) is the target format for all messages between the grid members, which transform their specific formats to the target format. Such transformations are defined as shells.

We envision architectural knowledge sharing in a grid-setting, an instantiation of the knowledge grid discussed in [21]. The basic idea is sketched in Figure 3. The model depicted in the center is the core model of what concerns architectural knowledge. Organization-specific models provide a specialization hereof. In the figure, *design principles* (a kind of Architectural Design Decision) are made available by RFA, while *quality criteria* (another kind of Architectural Design Decision) are made available by RDB (see also Section 6). Both organizations may offer visualization services (e.g. tabular versus graph-based) to visualize Architectural Design Decisions. Suppose a third organization, X, is looking for Architectural Design Decisions that are shared on the grid. Because of the specializations of the notion of an Architectural Design Decision at RFA and RDB, this query translates into a search for *design principles* at RFA, and *quality criteria* at RDB. Both results will be returned, even using the local visualization services of RFA and RDB.

We have started to use our core model to realize our envisioned grid-like setting to share architectural knowledge. We have built some simple services to process specific architectural knowledge: a word processor plug-in to annotate the rationale in architectural documents, and a service to visualize architectural design decisions using cluster-based browsing [4]. Future work includes construction of the grid service infrastructure based on the core model and specific shells, as well as new services to be shared among organizations participating in the grid.

9 Conclusions

In this paper we have presented a core model of architectural knowledge. This core model is the result of the execution of an action research cycle. Experimentation with an earlier version of the model identified mismatches between the model and industrial practice. Those mismatches have been overcome by ensuring the new version of our model of architectural knowledge is both complete and minimalistic. It is this latter feature that led us to adopt the term 'core model'. The validity of the claim of completeness of our core model is made plausible by an attempt to falsify this validity using various sources from literature.

During experimentation with the earlier version of the model, we identified four perspectives on architectural knowledge management in four different industrial organizations: sharing, compliance, discovery, and traceability. Subsequent application of the core model allowed us to identify probable causes and remedies for problems with architectural knowledge management encountered in those organizations: implicit relations between architectural decisions is the likely cause for the problems with sharing architectural knowledge a particular software development organization encounters; the compliance issues in a multi-site development organization are probably due to a decision making process that remains invisible to the affected parties; an organization that performs software product audits has to deal with the fact that, since the result of architectural decisions is not confined to a single product artifact, the relation between 'quality' and 'architecture' is often not obvious at first sight; long-term development projects in a scientific organization benefit from improved traceability if a better distinction is made between different concepts that play a role in the decision making process. We aim to alleviate those problems in a next iteration of our research with tools, methods, and techniques that address the probable causes of these problems with architectural knowledge management.

Acknowledgment

The authors would like to thank Rich Hilliard for constructive feedback on an earlier version of this paper. This research has been partially sponsored by the Dutch Joint Academic and Commercial Quality Research & Development (Jacquard) program on Software Engineering Research via contract 638.001.406 GRIFFIN: a GRId For inFormatIoN about architectural knowledge.

References

1. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 2nd edn. SEI Series in Software Engineering. Addison-Wesley Pearson Education, Boston (2003)
2. Ali Babar, M., Gorton, I., Jeffery, R.: Capturing and Using Software Architecture Knowledge for Architecture-Based Software Development. In: 5th International Conference on Quality Software (QSIC), Melbourne, Australia, pp. 169–176 (2005)
3. Bosch, J.: Software Architecture: The Next Step. In: Oquendo, F., Warboys, B., Morrison, R. (eds.) EWSA 2004. LNCS, vol. 3047, pp. 194–199. Springer, Heidelberg (2004)

4. Kruchten, P., Lago, P., van Vliet, H.: Building up and Reasoning about Architectural Knowledge. In: Hofmeister, C., Crnkovic, I., Reussner, R. (eds.) QoSA 2006. LNCS, vol. 4214, pp. 39–47. Springer, Heidelberg (2006)
5. Shaw, M., Clements, P.: The Golden Age of Software Architecture. IEEE Software 23(2), 31–39 (2006)
6. Tyree, J., Akerman, A.: Architecture Decisions: Demystifying Architecture. IEEE Software 22(2), 19–27 (2005)
7. van der Ven, J.S., Jansen, A., Nijhuis, J., Bosch, J.: Design decisions: The Bridge between Rationale and Architecture. In: Dutoit, A. (ed.) Rationale Management in Software Engineering, pp. 329–346. Springer, Heidelberg (2006)
8. Tang, A., Ali Babar, M., Gorton, I., Han, J.: A Survey of the Use and Documentation of Architecture Design Rationale. In: 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), Pittsburgh, USA, pp. 89–98 (2005)
9. Jansen, A., Bosch, J.: Software Architecture as a Set of Architectural Design Decisions. In: 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), Pittsburgh, USA, pp. 109–120 (2005)
10. Wang, Z., Sherdil, K., Madhavji, N.H.: ACCA: An Architecture-centric Concern Analysis Method. In: 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), Pittsburgh, USA, pp. 99–108 (2005)
11. Hofmeister, C., Kruchten, P., Nord, R.L., Obbink, H., Ran, A., America, P.: Generalizing a Model of Software Architecture Design from Five Industrial Approaches. In: 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), Pittsburgh, USA, pp. 77–86 (2005)
12. Akerman, A., Tyree, J.: Position on Ontology-Based Architecture. In: 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), Pittsburgh, USA, pp. 289–290 (2005)
13. Avison, D., Lau, F., Myers, M., Nielsen, P.A.: Action Research. Communications of the ACM 42(1), 94–97 (1999)
14. de Boer, R.C., Farenhorst, R., van der Ven, J.S., Clerc, V., Deckers, R., Lago, P., van Vliet, H.: Structuring Software Architecture Project Memories. In: 8th International Workshop on Learning Software Organizations (LSO), Rio de Janeiro, Brazil, pp. 39–47 (2006)
15. IEEE: IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. Standard 1471-2000, IEEE (2000)
16. Object Management Group: Software Process Engineering Metamodel Specification. Technical Report formal/05-01-06, Object Management Group (2005)
17. ISO/IEC: Software engineering - Product quality - Part 1: Quality model. Technical Report ISO/IEC 9126-1, ISO/IEC (2001)
18. Kruchten, P.: An Ontology of Architectural Design Decisions in Software-Intensive Systems. In: 2nd Groningen Workshop on Software Variability Management, Groningen, The Netherlands (2004)
19. Farenhorst, R., de Boer, R.C.: Core Concepts of an Ontology of Architectural Design Decisions. Technical Report IR-IMSE-002, Vrije Universiteit Amsterdam (2006)
20. Gorton, I.: Essential Software Architecture. Springer, Heidelberg (2006)
21. Zhuge, H.: The Knowledge Grid. World Scientific Publishing Co., Singapore (2004)