Using Architectural Decisions

Jan S. van der Ven, Anton Jansen, Paris Avgeriou, and Dieter K. Hammer
University of Groningen, Department of Mathematics and Computing Science,
PO Box 800, 9700AV Groningen, The Netherlands,
[salvador|anton|paris|dieter]@cs.rug.nl,
WWW home page: http://search.cs.rug.nl

***Abstract*— There are increasing demands for the explicit representation and subsequent sharing and usage of architectural decisions in the software architecting process. However, there is little known on how to use these architectural decisions, or what type of stakeholders need to use them. This paper presents a use case model that arose from industrial needs, and is meant to explore how these needs can be satisfied through the effective usage of architectural decisions by the relevant stakeholders. The use cases are currently being validated in practice through industrial case studies. As a result of this validation, we argue that the usage of architectural decisions by the corresponding stakeholders can enhance the quality of software architecture.**

I. Introduction

One of the proposed ways to advance the quality of software architecture is the treatment of architectural decisions [1], [2], [3], [4] as first-class entities and their explicit representation in the architectural documentation. From this point of view, a software system's architecture is no longer perceived as interacting components and connectors, but rather as a set of architectural decisions [5]. The main reason that this paradigm shift improves the quality of software architecture is that it reduces *Architectural Knowledge Vaporization* [1], [5]. It is presently not possible to completely eliminate vaporization, as the result still depends on the judgment and the chosen tradeoffs that the architect makes.

Architectural knowledge vaporizes because most of the *architectural decisions*, which are the most significant form of architectural knowledge [6], are lost during the development and evolution cycles. This is due to the fact that architectural decisions are neither documented in the architectural document, nor can they be explicitly derived from the architectural models. They merely exist in the form of tacit knowledge in the heads of architects or other stakeholders, and thus inevitably dissipate. The only way to resolve this problem is to grant architectural decisions first-class status and properly integrate them within the discipline of software architecture.

Although the domain of architectural decisions is receiving increasing attention by the software architecture community, there is little guidance as to how architectural decisions can be used during both architecting and in the general development process. In fact, in order to give architectural decisions first-class status there should be a systematic approach that can support their explicit documentation and usage by the architect and the rest of the stakeholders. We believe that it is too early at this stage to introduce methods or processes, and even more so supporting systems for creating and subsequently using architectural decisions. We argue that, first, we need to understand the complex nature of architectural decisions, and their role in the software development process and within an organization.

To achieve this goal, we present in this paper a use case model, that elaborates on two important issues: first, which are the stakeholders that need to use architectural decisions; second, how can the decisions be used by the relevant stakeholders. We have worked with industrial partners to understand the exact problems they face with respect to loss of architectural decisions. We have thus compiled a wish list from practitioners on the use of architectural decisions. Furthermore we have combined this wishlist with what we consider as the ideal requirements for a system that supports the usage of architectural decisions. We have thus followed both a bottom-down and a top-down approach and finally merged them into a use case model that represents real and ideal industrial needs for effective usage of architectural decisions. Finally, we validated this use case model in practice, by applying it at the industrial partners in small case studies.

The idea of a use case model for using architecture knowledge was introduced in [2], which forms the foundation work for our paper. This idea is further elaborated in [7]. It discusses the concept of architectural knowledge, from an ontological perspective, and typical usages of architectural knowledge in a broad context.

The rest of the paper is structured as follows: in Section 2 we give an overview of how our industrial partners defined the needs for using and sharing architectural decisions. Section 3 presents the use case model, including the actors and system boundary. The ongoing validation of the use cases is conducted in Section 4. Section 5 discusses related work in this field and Section 6 sums up with conclusions and future work.

II. From industrial needs to Use Cases

The use cases that are described in the rest of the paper refer to a potential system that would support the management of architectural decisions. To the best of our knowledge, there is no such system implemented yet. We envision this system as a Knowledge Grid [8]: "an intelligent and sustainable interconnection environment that enables people and machines to effectively capture, publish, share and manage knowledge resources".

Before pinpointing the specific requirements for a knowledge grid in the next sections, it is useful to consider the more

generic requirements by combining the areas of knowledge grids and architectural decisions. First, this system should support the effective collaboration of teams, problem solving, and decision making. It should also use ontologies to represent the complex nature of architectural decisions, as well as their dense inter-dependencies. Furthermore, it must effectively visualize architectural decisions and their relations from a number of different viewpoints, depending on the stakeholders' concerns. Finally, it must be integrated with the tools used by architects, as it must connect the architectural decisions to documents written in the various tools or design environments, and thus provide traceability between them.

We are currently participating in the Griffin project [9] that is working on tools, techniques and methods that will perform the various tasks needed for building this knowledge grid. Until now, the project has produced two main results: a use case model, and a domain model. The domain model describes the basic concepts for storing, sharing, and using architectural decisions and the relationships between those concepts [10]. The use case model describes the required usages of the envisioned knowledge grid. The use cases are expressed in terms of the domain model, in order to establish a direct link between the concepts relevant to architectural decisions (the domain model), and how the architectural decisions should be used (use cases). The focus in this paper is on the use case model.

Four different industrial partners participate in the Griffin project. They are all facing challenges associated to architectural knowledge vaporization. Although the companies are of different nature, they all are involved in constructing large software-intensive systems. They consider software architecture of paramount importance to their projects, and they all use highly sophisticated techniques for maintaining, sharing and assessing software architectures. Still, some challenges remain.

We conducted qualitative interviews with 14 employees of these industrial partners. Our goal was to analyze the problems they faced concerning sharing architectural knowledge, and to identify possible solutions to these problems. People with different roles were interviewed: architects (SA), project managers (PM), architecture reviewers (AR), and software engineers (SE). A questionaire (see the appendix at the end of this paper) was used to streamline and direct the interviews. The questionaire was not directly shown to the employees, but used as a starting point and checklist for the interviewers.

The results from the interviews were wrapped up in interview reports that described the current challenges and envisioned solutions by these companies. The interview reports contained some needs from the interviewees, which included:

1) Find relevant information in large architectural descriptions (SA, PM, AR).
2) Add architectural decisions, relate them to other architectural knowledge like architectural documentation, or requirement documentation (SA, PM).
3) Search architectural decisions and the underlying reasons, construct (multiple) views where the decisions are represented (SA, PM, AR).
4) Identify what knowledge should minimally be made available to let developers work effectively (SA).
5) Identify the changes in architectural documentation (PM).
6) Identify what architectural decisions have been made in the past, to avoid re-doing the decision process. This include identifying what alternatives were evaluated and the issues that played some critical role at that time (SA, PM, AR, SE).
7) Reuse architectural decisions (SA, PM, SE).
8) Keep architecture up-to-date during development and evolution (SA, PM).
9) Get an overview of the architecture (SA, PM, AR).

The interview reports and the needs stated in these reports form the starting point for constructing the use cases. Except for this bottom-up approach we also followed a top-down approach: we thought about the ideal usages of a system that supports the usage of architectural knowledge. This was necessary as most of the interviewees had a rather implicit and vague notion of architectural decisions and had not thought of using architectural decisions, represented as first-class entities. Both real needs from the interviewees and ideal needs proposed by the research group were merged into a use case model presented in the next section.

III. The Use Case Model

This section elaborates on a set of use cases that roughly define the requirements for a potential knowledge grid. First, we describe the actors of the knowledge grid, starting from the roles of our interviewees. After this, the primary actor and the scope are discussed. To understand the dependencies between the use cases a use case model consisting of 27 use cases, including the relations, is presented in figure 1. Besides presenting the dependencies among the use cases, the figure also relates the use cases to the identified needs described in the previous section. Note that this is not a one-to-one mapping; some needs resulted in multiple use cases and few use cases do not originate from needs but from 'ideal' requirements.

A. Actors

We identified the following actors being relevant for the use cases, based on the roles of the interviewees.

- *Architect.* Architects should be able to create and manage an architecture, and get an overview of the status of the architecture. This results in demands for views that show the coverage of requirements or describe the consistency of the design. Also, the architect is responsible for providing stakeholders with sufficient information, to ensure that their concerns are met in the architecture design.
- *Architecture Reviewer.* Architecture reviewers are often interested in a specific view on the architecture. They can be colleagues, experts from a certain field, or reviewers from an external organization. They want to understand the architecture quickly and want to identify potential pitfalls in the architecture, like poorly founded architectural

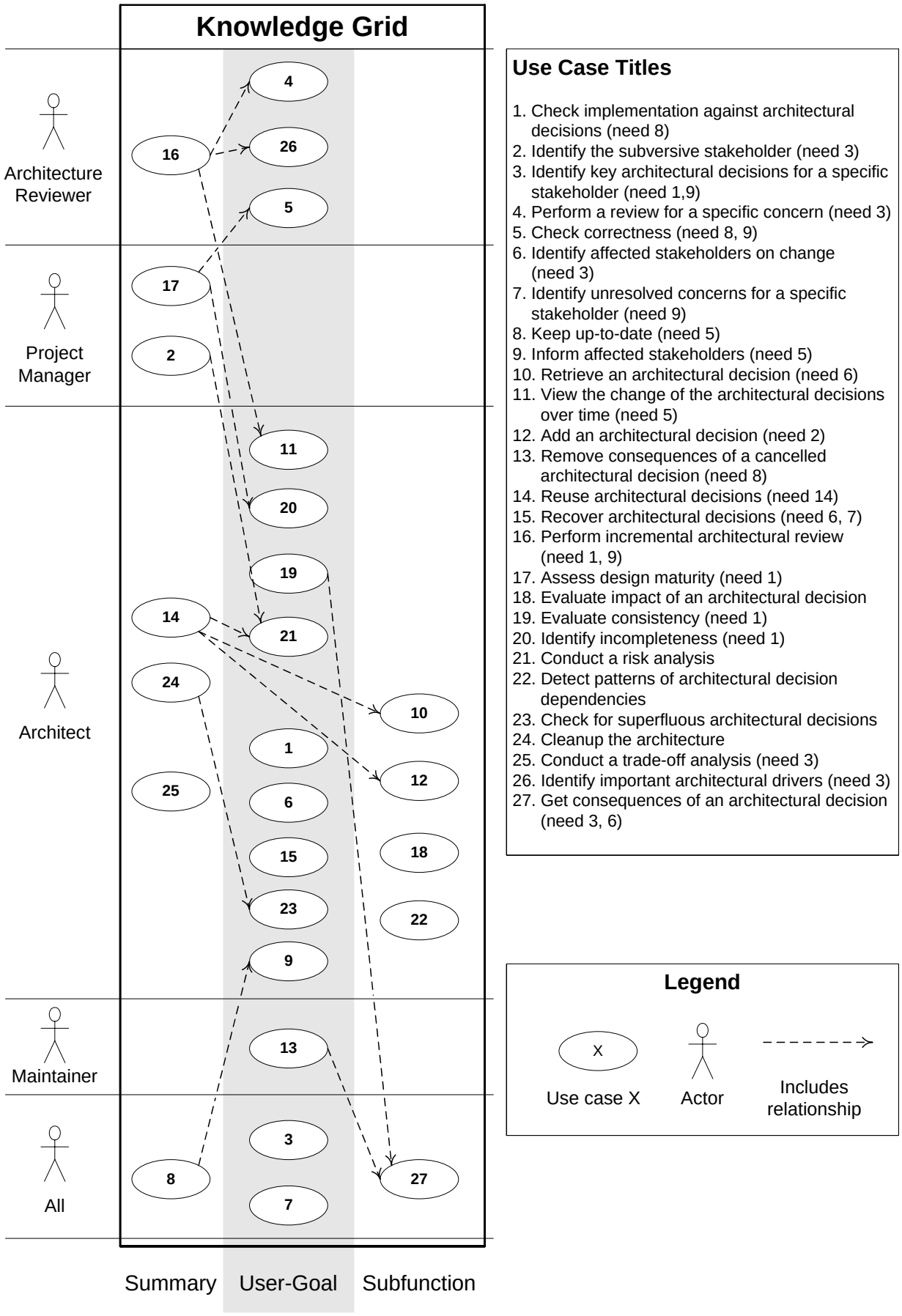

Fig. 1. Use case diagram

decisions, architectural incompleteness, or architectural inconsistency.

- *Project Manager.* The concerns of the project manager are usually driven by the planning; what is the status of the architecture, are there potential upcoming problems or risks, and how can we address them? The project manager also addresses people-related issues, e.g. which stakeholder is the biggest risk for the architecture?
- *Developer.* The primary concern of the developer is that the architecture should provide sufficient information for implementing the system. The descriptions must be unambiguous. Also, the developer must know where to look for the necessary knowledge; this can be in the architectural documentation, or by knowing which person to contact.
- *Maintainer.* The maintainer is often ignored as a stakeholder of an architecture. However, the maintainer is one of the most important actors when the architecture has to evolve. The maintainer has interest in the evolution of the architecture (up-to date information), and the consequences of changes in the architecture.

We encountered that the different companies used different terms for the roles they have in the software development process. The list of actors presented above is an abstraction of those different roles.

B. Describing the use cases

We present the use cases, as mandated in [11], using the following elements:

- *Scope.* All the use cases are defined as an interaction on a knowledge grid type of system (see section 2.1). From the use case model perspective, this system is considered a black-box system.
- *Goal level.* The descriptions from the interviews were very diverse in detail. As a consequence, some use cases describe a single interaction on the system (e.g. add an architectural decision), while others are very high-level demands of the system (e.g. perform an incremental architectural review). We adopted three goal levels from [11] of a decreasing abstraction: Summary, User-goal and Subfunction, for describing this difference. A Summary goal use case can involve multiple User-goals use cases, and often have a longer time span (hours, days). A User-goal use case involves a primary actor using the system (in Business Process Management often called elementary business process), often in one session of using the system. Subfunction use cases are required to carry out User-goal use cases. They typically represent an elementary action on the system, which is used by multiple User-goal use cases.
- *Primary actor.* The list of actors described in section III-A are used to determine the primary actor for a specific use case. Sometimes, a use case can be performed by all actors (e.g. identify key architectural decisions). In these cases, the term All is used as a substitute for the primary actor. In other cases, when the type of actor affects the use case, the most suitable actor was selected as primary actor, and the others were left out.
- *Main success scenario and steps.* First, a short description of the use case was constructed. From this, a set of steps was defined, describing the main success scenario. Due to space constraints, this is not shown for all the use cases. In the next section, four use cases are described in detail.
- *Includes relationships.* The "include" relationships between the use cases are based on the steps defined for these use cases. This relationship expresses that a use case contains behavior defined in another use case, as defined in UML 2.0 [12]. When a use case includes another use case with a different primary actor, this typically means that the first actor will ask the second actor to perform the specified use case. For example, in use case 2 (Identify the subversive stakeholder), the project manager will ask the architect to conduct a risk analysis (use case 21). Off course one person can also have multiple roles, and thus perform as the primary actor in both use cases.

Figure 1 presents the characteristics (Primary actor, goal level, and name) of the use case model, which consists of 27 use cases. Note that to enhance the readability, the uses relationship (between the actor and the use case) is not visualized with arrows, but by horizontal alignment. For example, the architecture reviewer acts as a primary actor for use cases 16, 4, 26, and 5. The use cases are vertically divided in the three goal levels: Summary, User-goal and Subfunction. For example, use case 16 is a Summary use-case and use case 4 an User-goal.

IV. Use Case Validation

A use case model like the one presented in section III can not be directly validated in a formal, mathematical sense. Instead, the use cases need to be applied in practice and their effect on the development process should be evaluated. However, before the use cases can be applied, the use cases need further refinement to become usefull. In this validation section, we present these refinements, demonstrate the relevance of the use cases in an industrial setting, and present the improvement these use-cases have made on the development process.

Currently, the Griffin project is involved in conducting case studies at our industrial partners to validate the use cases. In this section we briefly present the Astron Foundation case study. Astron is currently engaged in the development of the LOw Frequency ARray (LOFAR) for radio astronomy [13]. LOFAR pioneers the next generation of radio telescope and will be the most sensitive radio observatory in the world. It uses many inexpensive antennas combined with software, instead of huge parabolic dishes, to observe the sky. This makes LOFAR a software intensive telescope. LOFAR will consists of around 15.000 antenna's distributed over 77 different stations. Each antenna will generate around 2 Gbps of raw data. The challenge for LOFAR is to communicate and process the resulting 30Tbps data stream in real-time for interested scientists.

In the LOFAR system, architectural decisions need to be shared and used over a time span of over 25 years. This is due to the long development time (more then 10 years), and a required operational lifetime of at least 15 years. Astron is judged by external reviewers on the quality of the architecture. The outcome of these reviews influences the funding, and consequently the continuation of the project. Therefore, it is evident that the architecture has to hold high quality standards.

Together with Astron, we identified eight use cases being of primary concern for the LOFAR case study: 5, 7, 10, 12, 15, 17, 19, and 20. This section focuses on assessing the design maturity, which is a major concern for Astron. Assessing the design maturity is a specialization of the earlier identified need for getting an overview of the architecture (see section II, need 9). The following use-cases are relevant with regard to this assessment:

- Asses design maturity (UC 17, see figure 2)
- Identify incompleteness (UC 20, see figure 3)
- Check correctness (UC 5, see figure 4)
- Evaluate consistency (UC 19, see figure 5)

Of these four use cases, use case 17 is the only *Summary level* use case (see figure 1). Use cases 5, 19, and 20 are used by this use case. In the remainder of this section, these use cases are presented in more detail. For each use case, the following is presented: the relevance to Astron, the current practice at Astron, a more elaborate description of the use case, and the envisioned concrete realization within Astron.

The elaborated use case descriptions make use of the concept of **knowledge entity**. All the domain concepts defined within the knowledge grid are assumed as being knowledge entities. For this case study, this includes among others: architectural decisions, rationales, decision topics, alternatives, requirements, specifications, assumptions, rules, constraints, risks, artifacts, and the relationships among them.

A. UC 17: Assess design maturity

Relevance The design maturity is an important part of the quality of the LOFAR architecture. LOFAR is constructed using cutting edge technology to enquire maximum performance. However, due to the long development time, these cutting edge technologies are typically emerging when the initial architecture design is being made. So, the architecture has to be made with components that do not yet exist. It is therefore hard for Astron to make a judgment whether the architecture is sufficiently matured to start actual construction.
Current Practice Within the LOFAR case study, the design maturity is first assessed for the various subsystems by each responsible architect. For each subsystem the main issues with regard to incompleteness, correctness, and consistency are reported. Based on these reports, the opinions of the architects and project management it is decided whether the system is mature enough to be proposed to the external reviewers to proceed to the next project phase.
Use case realization The design maturity use case is presented in figure 2. This use case consists of three other use cases that in turn are used to check the architecture for completeness, correctness, and consistency. These use cases are presented in the remainder of this section.

UC 17: Assess design maturity

Description: This use case verifies whether a system conforming to the architecture can be made or bought. The architect wants to know when the architecture can be considered as finished, complete, and consistent.
Primary actor: Project Manager.
Scope: Knowledge grid
Level: Summary.
Precondition: None.
Postcondition: The knowledge grid provides an overview of the matureness, and reports potential risks.
Main success scenario:

1) Identify incompleteness (UC 20)
2) Check correctness (UC 5)
3) Evaluate consistency (UC 19)
4) The grid generates a report based on the knowledge of the previous steps

Extensions: None

Fig. 2. Use case 17

B. Use Case 20: Identify incompleteness

Relevance Use case 20 determines whether the architecture covers all (essential) requirements. For Astron this is relevant from a management perspective; incompleteness gives pointers to directions where additional effort should be concentrated.
Current practice Astron checks for the completeness of the architecture description by peer review and risk assessment. The peer review is done iteratively; fellow architects give feedback (among others) on the completeness of the architectural descriptions. A risk assessment is performed before every external review. The result of this process, a risk matrix, is used for the next iteration of the architectural description. During the design phase, the architect signifies specific points of incompleteness, typically by putting keywords like ‘tbd’ (to be determined), or ‘tbw’ (to be written) in the architecture documents.

For example, in the central processor, the signals of the antenna’s should be correlated with each other. Therefore, the signals of all the stations should be routed all-to-all. However, the architectural decision on what network topology to use for this task is still incomplete, as some alternatives have been evaluated, but no suitable (cost-effective) solution can be selected so far.
Use case realization The general use case is described in figure 3. For Astron this is realized by the following:

Risks Currently, the relationships between the identified risks (for example in the risk matrix) and the design (in the architectural documentation) are not explicitly clear. The knowledge grid allows the architect to relate risks to particular parts of the design and to architectural decisions. This use case enables the architect to *partially* check the completeness of

Use Case 20: Identify incompleteness

Description: In this use case, the system provides a report about the structure of the architectural decisions.
Primary actor: Architect.
Scope: Knowledge grid
Level: User-goal.
Precondition: The user is known within the knowledge grid.
Postcondition: The knowledge provides an overview of the incomplete knowledge entities.
Main success scenario:

1) The architect selects a part of the architecture.
2) The knowledge grid identifies the knowledge entities in the part.
3) The grid reports about the incompleteness of these knowledge entities.

Extensions: None

Fig. 3. Use case 20

the mitigation of risks, as every risk should be addressed by at least one architectural decision. Whether the risk is actually addressed by the decision, is checked by UC 5, presented in the next subsection.

Requirements As an example of inconsistency indicators every requirement should lead to one or more (mostly non-formal, usually textual) specifications. It can thus be automatically determined which requirements are not covered by any specifications.

Visualization Possibilities of visualization for incompleteness can be visualized by a "to-do" list of open decision topics, or visual indicators in the documentation (e.g. icons, or coloring of text pieces).

C. Use Case 5: Check correctness

Relevance Besides completeness, it is also important to know whether the architectural decisions actually address the requirements. In this sense, correctness is complimentary to completeness. For example, completeness only means that there exist decisions taken with respect to all requirements, while correctness means that these decisions actually lead to a solution that meets these requirements. Astron spends considerable effort in verifying the correctness of the design. Prototypes of major hardware and software components are made and evaluated. Simulations and models are used as well. For example, to deal with the major concern of the enormous amounts data to be processed, Astron has developed an elaborate performance model. This model allows the architects to simulate and validate the correctness of many different concepts for distributed data processing.
Current practice Similar to the check for completeness, peer reviews are used to verify the correctness of the design description. Domain experts verify the design description created by the architect. Based on this feedback, the architect adapts the design description. Doubts about the correctness of parts of the design are typically annotated with the key word 'tbc' (to be confirmed) or placed in a separate open issue sections. If there is any doubt about the *way* in which the correctness is verified, keywords like 'under discussion' are typically used in the architectural documentation.

Use Case 5: Check correctness

Description: In this use case, the knowledge grid supports the user in validating the correctness of the architectural decisions addressing the requirements.
Primary actor: Architect.
Scope: Knowledge grid
Level: Subfunction.
Precondition: The knowledge grid contains incompleteness information of the design.
Postcondition: The knowledge grid contains markings about the correctness; An overview of incorrect knowledge entities is provided.
Main success scenario:

1) The architect selects a set of requirements in the knowledge grid.
2) The knowledge grid provides a list of related architectural decisions and other related knowledge entities (e.g. assumptions, rules, constraints).
3) The architect evaluates related elements and marks the incorrect elements.
4) The architect continues with the next requirement.
5) The knowledge grid provides an overview of incorrect architectural decisions and requirements.

Extensions:
3a. The elements are correct, the architect marks them as such.

Fig. 4. Use case 5

For example, there has been an incorrect assessment of the distributed behavior of the used calibration algorithm. It was expected that each node used 80% local data, and that for the remaining 20% all the data on the other nodes was needed. Based on this assessment the architectural decision was made to use a distributed database grid. However, during performance tests it turned out that for this 20% the data of only one or two other nodes was needed, instead of *all* the other nodes. Consequently, the architectural decision turned out to be wrong, as the architectural decision for a centralized database is a significant better alternative. In retrospect, verification of the architectural decision by the correct domain expert could have prevented this situation from arising in the first place.
Use Case realization The knowledge grid itself cannot determine the correctness of the architectural decisions without in-depth semantic knowledge of the underlying architectural models. Therefore, this use case makes provision for assisting the architect in determining the correctness of the design, rather than that the knowledge grid determines the correctness itself.

Requirements For each requirement or risk, the architect needs to find out whether the involved architectural decisions correctly address the requirement or risk. This use case describes how this process can be supported.

Visualize The visualization of the incompleteness can subsequently be used to visualize incorrect elements. However, since the checking of correctness is mostly manual job for the architect, the results may vary when different people are checking the correctness. Integration of this information is then needed.

D. Use Case 19: Evaluate consistency

Relevance This use case is concerned with the consistency between the architectural decisions themselves. As the LOFAR project consists of many components that are developed in parallel, detecting contradictions is important, as this provides an early warning for mistakes in the overall design. Inconsistencies make the design of the system harder to understand and create problems during the realization of the system.

Current practice Checking for inconsistencies in textual descriptions is largely a manual job. The part of the design that is modeled (e.g. in the performance models) can automatically be checked for inconsistencies. However, they only cover a very small part of the overall design, and therefore a small part of the architectural decisions. Most of the inconsistencies are found by inspection, either by the architect or reviewer.

For example, there has been an inconsistency in LOFAR between the protocol used by the central processor (the correlator of the radio signals) and the stations (the locations where the antenna's are residing). Although large efforts have been put in a consistent definition of the data packet header, versioning etc., the used definition of how to interpret the subband data turned out to be inconsistent. For the station a subband was defined starting with the lowest frequency leading to the highest frequency of the subband, while for the central processor it was defined the other way around.

Use case realization The architect is supported with relevant context information in the decision making process. For Astron, this will include the visualization of relevant requirements, and closely related architectural decisions and specifications. Techniques similar to the work of [14] could be used for this. Furthermore, once an inconsistency is detected, the architect is supported with a visualization of the relevant architectural decisions. This allows the architect not only to confirm an inconsistency, but also to detect its cause and consequently resolve it.

Use Case 19: Evaluate consistency

Description: In this use case, the knowledge grid supports the user in detecting inconsistencies in the architecture design.
Primary actor: Architect.
Scope: Knowledge grid.
Level: User-goal.
Precondition: The user is known within the knowledge grid.
Postcondition: The knowledge grid contains markings about the consistency; An overview of inconsistent knowledge entities is provided.
Main success scenario:

1) The architect selects a subset of architectural knowledge in the grid.
2) Architect selects a specific knowledge entity or a part of the design, and asks the knowledge grid for consistency assistance.
3) The knowledge grid provides a list of related (and potentially inconsistent) knowledge entities.
4) The architect marks the inconsistent knowledge entities.
5) The architect repeats steps 3 and 4 for the remaining knowledge entities.
6) The knowledge grid provides an overview of inconsistent knowledge.

Extensions:
4a. The knowledge entities are consistent, the architect marks them as such.

Fig. 5. Use case 19

V. Related Work

Software architecture design methods [15], [16] focus on describing how sound architectural decisions can be made. Architecture assessment methods, like ATAM [15], assess the quality attributes of a software architecture. The use cases presented in this paper describe some assessment scenarios that could be reused from these design and assessment methods.

Software documentation approaches [17], [18] provide guidelines for the documentation of software architectures. However, these approaches do not explicitly capture the way to take architectural decisions and the rationale behind those decisions. The presented use cases describe how stakeholders would like work with this knowledge.

Architectural Description Languages (ADLs) [19] do not capture the decisions making process in software architecting either. An exception is formed by the architectural change management tool Mae [20], which tracks changes of elements in an architectural model using a revision management system. However, this approach lacks the notion of architectural decisions and does not capture the considered alternatives or rationales, something the knowledge grid does.

Architectural styles and patterns [21], [22] describe common (collections of) architectural decisions, with known benefits and drawbacks. Tactics [15] are similar, as they provide clues and hints about what kind of techniques can help in certain situations. Use case 22 (Detect patterns of architectural decision dependencies), can be used to find these kinds of decisions.

Currently, there is more attention in the software architecture community for the decisions behind the architectural model. Tyree and Akerman [3] provide a first approach on documenting design decisions for software architectures. Concepts and guidelines for explicit representations of architectural decisions can be found in the work of Babar et al. [23] and

our own work [5], [6]. Closely related to this is the work of Lago and van Vliet [24]. They model assumptions on which architectural decisions are often based, but not the architectural decisions themselves. Kruchten et al. [2], stress the importance of architectural decisions, and show classifications of architectural decisions and the relationship between them. They define some rough outlines for the use cases for describing how to use architectural knowledge. Furthermore, they provide an ontology based visualization of the knowledge in the grid. We emphasize more on the explicit modeling of the use cases and are validating a set of extended use cases in the context of a case study.

Integration of rationale and design is done in the field of design rationale. SEURAT [25] maintains rationales in a RationaleExplorer, which is loosely coupled to the source code. These rationales have been transferred to the design tool, to let the rationales of the architecture and implementation level be maintained correctly. DRPG [26] couples rationale of well-known design patterns with elements in a Java implementation. Just like SEURAT, DRPG also depends on the fact that the rationale of the design patterns is added to the system in advance. The importance of having support for design rationales was emphasized by the survey conducted by Tang et al. [4]. The results emphasized the current lack of good tool support for managing design rationales. The use cases presented in this paper are an excellent start for requirements for such tools.

From the knowledge management perspective, a web based tool for managing architectural knowledge is presented in [23]. This approach uses tasks to describe the use of architectural knowledge. These tasks are much more abstract then the use cases defined in this paper (e.g. architectural knowledge use, architectural knowledge distribution).

Finally, another relevant approach is the investigation of the traceability from the architecture to the requirements [27]. Wang uses Concern Traceability maps to reengineer the relationships between the requirements, and to identify the root causes. The results from such systems could be valuable input for defining the relationships between knowledge entities, as used in our validation.

VI. Conclusions and Future Work

In order to upgrade the status of architectural decisions, we must first understand how they can be shared and used by a software development organization. For this purpose, we have proposed a use case model that came out of industrial needs and aims to fill specific gaps and in particular to alleviate the dissipation of architectural decisions. This use case model is considered as the black-box view of a knowledge grid type of system that is envisioned to enrich the architecting process with architectural decisions.

A reasonable question to reflect upon is: how exactly was the software architecture quality enhanced by the use case model proposed in this paper? Although pinpointing what exactly constitutes the quality of software architecture per se is a difficult issue, we can identify five arguments in this case:

- **Less expensive system evolution**. As the systems need to change in order to deal with new requirements, new architectural decisions need to be taken. Adding, removing and modifying architectural decisions can be based on the documentation of existing architectural decisions that reflect the original intent of the architects. Moreover, architects may be less tempted to violate or override existing decisions, and they cannot neglect to remove them. In other words the architectural decisions are enforced during evolution and the problem of *architectural erosion* [28] is reduced.
- **Enhanced stakeholder communication**. The stakeholders come from different backgrounds and have different concerns that the architecture document must address. Architectural decisions may serve the role of explaining the rationale behind the architecture to all stakeholders. Furthermore, the explicit documentation of architectural decisions makes it more effective to share them among the stakeholders, and subsequently perform tradeoffs, resolve conflicts, and set common goals.
- **Improved intrinsic characteristics of the architecture**. These concern attributes of the architecture, such as conceptual integrity, correctness, completeness and buildability [15]. Architectural decisions can support the development team to upgrade such attributes, because they give more complete knowledge, they provide a clearer and bigger picture. In other words, architectural decisions provide much richer input to the formal (or less formal) methods that will be used to evaluate these attributes.
- **Extended architectural reusability**. Reuse of architectural artifacts, such as components and connectors, can be more effectively performed when the architectural decisions are explicitly documented in the architecture. To reuse architectural artifacts, we need to know why they were chosen, what their alternatives were, and what benefits and liabilities they bring about. Such kind of reusability prevents the architects from re-making past mistakes or making new mistakes. Finally architectural decisions per se, can and should be reused, probably after slight modifications.
- **Extended traceability between requirements and architectural models**. Architectural decisions realize requirements (or stakeholders' concerns) on the one hand, and result in architectural models on the other hand. Therefore, architectural decisions are the missing link between requirements and architectural models and provide a two-way traceability between them [6]. The architect and other stakeholders can thus reason which requirements are satisfied by a specific part of the system, and vice-versa, which part of the system realizes specific requirements.

We are currently trying to validate the use case model in four industrial case studies to better understand the pragmatic industrial needs and make the use case model as relevant and

effective as possible. After this validation, we plan to perform a second iteration of validation interviews with the original interviewees from the first iteration, as well as more stakeholders with different roles, in order to fully cover the most significant roles. Furthermore external architects will also be asked to validate the use case model. In the meantime we have already attempted to implement parts of the knowledge grid in the form of tool support, which is used in the aforementioned case study of the Astron Foundation.

ACKNOWLEDGEMENTS

This research has partially been 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] J. Bosch, "Software architecture: The next step," in *Software Architecture, First European Workshop (EWSA)*, ser. LNCS, vol. 3047. Springer, May 2004, pp. 194–199.
[2] P. Kruchten, P. Lago, H. van Vliet, and T. Wolf, "Building up and exploiting architectural knowledge," in *WICSA 5*, November 2005.
[3] J. Tyree and A. Akerman, "Architecture decisions: Demystifying architecure," *IEEE Software*, vol. 22, no. 2, pp. 19–27, 2005.
[4] A. Tang, M. A. Babar, I. Gorton, and J. Han, "A survey of the use and documentation of architecture design rationale," in *Proceedings of WICSA 5*, November 2005.
[5] A. G. J. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *Proceedings of WICSA 5*, November 2005, pp. 109–119.
[6] J. S. van der Ven, A. G. J. Jansen, J. A. G. Nijhuis, and J. Bosch, "Design decisions: The bridge between rationale and architecture," in *Rationale Management in Software Engineering*, A. H. D. et al., Ed. Springer-Verlag, march 2006, ch. 16, pp. 329–348.
[7] P. Kruchten, P. Lago, and H. van Vliet, "Building up and reasoning about architectural knowledge," in *Proceedings of the Second International Conference on the Quality if Software Architectures (QoSA 2006)*, June 2006.
[8] H. Zhuge, *The Knowledge Grid*. World Scientific Publishing Company, 2004.
[9] Griffin project website, http://griffin.cs.vu.nl.
[10] R. Farenhorst, R. C. de Boer, R. Deckers, P. Lago, and H. van Vliet, "What's in a domain model for sharing architectural knowledge?" in *Proceedings of the 18th International Conference on Software Engineering and Knowledge Engineering (SEKE2006)*, July 2006.
[11] A. Cockburn, *Writing Effective Use Cases*. Addison Wesley, 2001.
[12] The Unified Modeling Language (UML) website, http://www.uml.org/.
[13] Lofar project website, http://www.lofar.org/.
[14] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Improving after-the-fact tracing and mapping: Supporting software quality predictions," *IEEE Software*, vol. 22, no. 6, pp. 30–37, November/December 2005.
[15] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice 2nd ed.* Addison Wesley, 2003.
[16] J. Bosch, *Design & Use of Software Architectures, Adopting and evolving a product-line approach*. ACM Press/Addison Wesley, 2000.
[17] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, *Documenting Software Architectures, Views and Beyond*. Addison Wesley, 2002.
[18] C. Hofmeister, R. Nord, and D. Soni, *Applied software architecture*. Addison Wesley, 2000.
[19] N. Medvidovic and R. N. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE Transactions on Software Engineering*, vol. 26, no. 1, pp. 70–93, 2000.
[20] A. van der Hoek, M. Mikic-Rakic, R. Roshandel, and N. Medvidovic, "Taming architectural evolution," in *Proceedings of the 8th European software engineering conference*. ACM Press, 2001, pp. 1–10.
[21] M. Shaw and D. Garlan, *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, Inc., 1996.
[22] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *A system of patterns*. John Wiley & Sons, Inc., 1996.
[23] I. G. Muhammad Ali Babar and R. Jeffery, "Toward a framework for capturing and using architecture design knowledge," University of New South Wales, Australia ans National ICT Australia Ltd., Tech. Rep. UNSW-CSE-TR-0513, june 2005.
[24] P. Lago and H. van Vliet, "Explicit assumptions enrich architectural models," in *ICSE '05: Proceedings of the 27th international conference on Software engineering*. New York, NY, USA: ACM Press, 2005, pp. 206–214.
[25] J. E. Burge and D. C. Brown, "An integrated approach for software design checking using design rationale," in *1st International Conference on Design Computing and Cognition (DCC '04)*, July 2004, pp. 557–576.
[26] E. L. A. Baniassad, G. C. Murphy, and C. Schwanninger, "Design pattern rationale graphs: Linking design to source," in *Proceedings of the 25th ICSE*, May 2003, pp. 352–362.
[27] Z. Wang, K. Sherdil, and N. H. Madhavji, "ACCA: An architecture-centric concern analysis method," in *5th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, November 2005.
[28] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *ACM SIGSOFT Software Engineering Notes*, vol. 17, no. 4, pp. 40–52, 1992.

APPENDIX
GRIFFIN QUESTIONNAIRE

This appendix contains the questionnaire that was used during the interviews with the stakeholders at the industrial partners. The questionnaire was sent to most interviewees in advance and used by the Griffin research team to see whether all relevant subjects have been discussed during the interview.

Introduction of yourself

1) Can you describe your role and responsibilities within the organization?
2) Can you give an estimate on what percentage of your time is spent on activities related to architecture? Examples include capturing architectural knowledge, communicating architectural knowledge to stakeholders, et cetera.
3) With what kind of stakeholders in the architecting process do you interact most?

Architecture

1) For the sake of clarity: what is your definition of "(software) architecture"? Does this definition differ from the generally accepted definition within your organization?
2) Can you describe the software design process, and the place the architecture takes in it?
3) Are architectures kept up-to-date during evolution? What techniques are used in keeping the architectures up-to-date?
4) For what stakeholders are architecture documents created? What are (generally spoken) the most important stakeholders?
5) Are tools, methods, templates, or architectural description languages (ADLs) used in constructing an architecture?
6) Are you satisfied about the way these tools, methods, templates, or ADLs are being utilized during the architecture construction process? Can you mention any improvement points?

Architectural knowledge

1) What architectural decisions are documented, and how?
2) Can you quantify the impact of Architectural Knowledge that is lost / not present / too implicit? Can you give examples?
3) Could you provide a top-3 list of burdens in modelling architectures? What are your ideas on this?

Your architectures of today and in the past

1) What are the most important quality characteristics of your architecture (or architectures)?
2) What kind of solutions do you provide in your design? Do you reuse certain solutions (e.g. architectural patterns) in your architectures?
3) (If possible to mention commonalities): With what kinds of design aspects do you deal explicitly in your architectures? Examples of design aspects include interfaces, error handling, execution architecture, data consistency, and robustness.
4) From what sources do you obtain information for these design aspects?
5) Is there a topic on which you foresee a big change in the use of architectures in the future?

A. Architecting in daily practice

1) How is the availability of architectural information planned, managed, and reviewed?
2) What will be (in your opinion) a big change in the architect's job in the future?
3) Looking back on the last few years, what would you reckon as a significant step forward in architecting support?
4) How would you prepare for this?
5) How is this change planned?