17 Collaborative Software Architecting through Knowledge Sharing

Peng Liang, Anton Jansen, Paris Avgeriou

Abstract: In the field of software architecture, there has been a paradigm shift from describing the outcome of the architecting process to documenting architectural knowledge, such as design decisions and rationale. Moreover, in a global, distributed setting, software architecting is essentially a collaborative process in which sharing and reusing architectural knowledge is a crucial and indispensible part. Although the importance of architectural knowledge has been recognized for a considerable period of time, there is still no systematic process emphasizing the use of architectural knowledge in a collaborative context. In this chapter, we present a two-part solution to this problem: a collaborative architecting process based on architectural knowledge and an accompanying tool suite that demonstrates one way to support the process.

Keywords: software architecture, collaborative architecting, architecting process, architectural knowledge, knowledge sharing, collaborative software engineering

17.1 Introduction

According to a recent paradigm shift in the field of software architecture [1][3][5], the product of the architecting process is no longer only the models in the various architecture views, but the broader notion of Architectural Knowledge (AK) [23]: the architecture design as well as the design decisions, rationale, assumptions, context, and other factors that together determine architecture solutions. Architectural (design) decisions are an important type of AK, as they form the basis underlying a software architecture [19]. Other types of AK include concepts from architectural design (e.g. components, connectors) [25], requirements engineering (e.g. risks, concerns, requirements), people (e.g. stakeholders, organization structures, roles), and the development process (e.g. activities) [10].

The entire set of AK needs to be iteratively produced, shared, and consumed during the whole architecture lifecycle by a number of different stakeholders as effectively as possible. The stakeholders in architecture may belong to the same or different organization and include roles such as: architects, requirements engineers, developers, maintainers, testers, end users, and managers etc. Each of the stakeholders has his/her own area of

expertise and a set of concerns in a system being developed, maintained or evolved. The architect needs to facilitate the collaboration between the stakeholders, provide AK through a common language for communication and negotiation, and eventually make the necessary design decisions and trade-offs.

However, in practice, there are several issues that hinder the effective stakeholder collaboration during the architecting process, which diminishes the quality of the resulting product. One of these problems is the lack of integration of the various architectural activities and their corresponding artifacts across the architecture lifecycle [17]. The different stakeholders typically have different backgrounds, perform discrete architectural activities in a rather isolated manner, and use their own AK domain models and suite of preferred tools. The result is a mosaic of activities and artifacts rather than a uniform process and a solid product.

This chapter focuses on how to integrate stakeholder-specific approaches and tools related to the individual architecting activities. We propose a two-part solution to this problem: a process and an accompanying tool suite. The first part integrates requirements engineering (RE) and the various architecting activities (e.g. analysis, synthesis, evaluation, maintenance etc.) and their consumed and produced AK, as well as the related stakeholders, into a single process model based on the principle of sharing AK. Note, that we have decided to take RE into account: even though it is technically not part of the architecting process, they are closely intertwined and affect one another [5].

The second part is the Knowledge Architect tool suite that supports the collaborative architecting process by realizing and integrating different tools that correspond to the various activities of the process. The tool suite demonstrates one way to support the process, which is derived from the requirements of our industrial partner; there are other ways to support the same activities depending on the organization, the domain, and the specific project at hand. Currently, the tool suite consists of the following tools: the Document Knowledge Client supporting architects writing an architecture document; the Excel and Python Plug-ins supporting system analysts performing quantitative architectural analysis; the Knowledge Repository acting as the central location to store all the relevant AK; the Knowledge Explorer allowing other stakeholders to search, inspect, and trace AK; the Knowledge Translator translating AK from one language to the other for easy understanding. An important feature of the tool suite is that the individual tools share their AK for specific activities through a central knowledge repository, thus providing traceability of the AK and automated checking across a wide range of architecting activities.

Sect. 17.2 of this chapter discusses collaboration in software architecting and the role of AK. Sect. 17.3 presents the integrated process for collaborative architecting, while Sect. 17.4 introduces the accompanying tool suite. Sect. 17.5 elaborates on the details of collaboration by applying the process and tooling, exemplified through a running example. The paper ends with a discussion on related work, followed by conclusions and directions for future work.

17.2 Theoretical Background

17.2.1 Collaboration in Software Architecting

Architecting is an inherently collaborative process between architects and several stakeholders, who have various concerns and viewpoints. Software architecture:

- allows stakeholders to work together, communicate, negotiate, and eventually agree upon the architectural decisions and rationale [34].
- defines the partition of both the technical architecture and the organizational teams building the system [5].
- resolves errors and deals with risks throughout the system [5].
- documents the explicit AK of the organization and the project to facilitate future evolution [1].

In [17], the authors of five industrial architecture design methods propose a common model for architecting, comprised of three fundamental architecting activities: architectural analysis, synthesis, and evaluation. They identify the problem of lack of integration between these activities and their corresponding artifacts and they propose to deal with this problem through the concept of a *backlog*: a collection of needs, issues, problems, ideas, which binds the 3 architecting activities together. Therefore, the backlog acts as a central knowledge artifact that is both produced and consumed by the 3 activities, facilitating their integration. In a collaborative setting, this integration problem is aggravated due to the distribution of stakeholders who have different backgrounds and expertise. In our approach, we also propose knowledge sharing as a promising solution, but at a larger scale: an elaborate set of AK is shared and reused across the proposed architecting process. The shared AK provides a common language for the distributed stakeholders to communicate, reason, and ensure their concerns are being addressed.

The general goals of collaboration in software engineering identified in [36] include: "*Driving convergence towards a final architecture and design*", "*Managing dependencies among activities, artifacts, and organizations*", "*Identifying, recording and resolving errors" and "Recording organizational memory*". We specialize these goals for collaboration in architecting and restate them as follows:

- producing an integrated and consistent architecture document that has emerged from iterative stakeholder negotiation and agreements.
- managing the dependencies and establishing traceability among architecting activities, artifacts and involved stakeholders.
- identifying, recording and resolving architectural conflicts, risks, inconsistency, and incompleteness.
- recording the knowledge which is relevant to the whole architecting process.

To evaluate how the proposed process and tool achieve these goals, we revisit them in the Conclusions section.

17.2.2 Knowledge Management for Collaborative Architecting

A distinction is often made in KM between two types of knowledge [30]: *tacit* (personalized) knowledge that resides in people's head, versus *explicit* knowledge that is codified in some form. The latter is often further characterized as documented or formalized knowledge. Documented knowledge is expressed in natural languages or drawings, e.g. Word and Excel documents that contain architecture description and analysis models. Formal knowledge is expressed (or annotated) in formal languages or models with clearly specified semantics. Typical examples of this form include AK ontologies [22] or AK domain models [25][10][2][9][20][21][36] that formally define concepts and relationships (e.g. *Design Decision* related to *Concern*). They aim at providing a common language for unambiguous interpretation by stakeholders. Formal AK can better facilitate activities for architectural collaboration than documented or tacit AK [10]. However, formal AK entails additional cost and effort [8].

Based on the knowledge types, Hansen et al. classify KM in two strategies [16]: *codification* aims at codifying knowledge and making it available for anyone through knowledge repositories; *personalization*, helps people to communicate knowledge instead of storing it. Both KM strategies are employed in software engineering activities [33]: most research and industry practice has been associated with codification [11], while personalization has been given less attention. In this chapter, we mainly focus

on codified AK in collaborative architecting. Personalization is also valuable, and will be further investigated in our future work.

17.3 A Process for Collaborative Software Architecting

The architecting process involves several stakeholders due to its cross-cutting nature from requirements to implementation. For large projects, several teams may work simultaneously on different parts or in different development stages of the whole system, and exchange information. AK is the most important part of the exchanged information and is of paramount importance to the architecting process.

To investigate the role of AK in the architecting process, we have closely cooperated with our industrial partner, Astron (the Dutch radio astronomy institute), which develops large and complex software systems for radio telescopes. What makes these systems interesting from a collaborative AK perspective is: (1) the development consortium consisting of multiple international partners, (2) the long development time of nearly a decade, (3) the long required operational lifetime of at least 20 years.
In this context, we first identified and described the requirements to manage AK in the architecting process of Astron through a number of use cases using our earlier work [37]. We subsequently identified the AK needed to execute these use cases and expressed this knowledge in a domain model [20][18]. Using both the domain and the use cases, we derived and generalized a collaborative architecting process that integrates the different architecting activities. To support this general process within Astron, we developed a tool suite, which is presented in Sect. 17.4.

Fig. 17.1 illustrates this derived process in terms of activities and AK produced and consumed. Furthermore, it visualizes the close interaction between architecture (solution space) and requirements (problem space), as they are closely intertwined [31]. Every architecting activity can provide feedback to the RE activity, as new insights, acquired during architecting, lead to a better understanding of the problem domain. It is noted that the AK-based architecting process is not sequential, but highly iterative and incremental: achieving an acceptable architecture requires an iterative design and evaluation process that allows refinement to address new requirements and trade-offs. The architecting activities and the related RE activity are briefly described as follows:

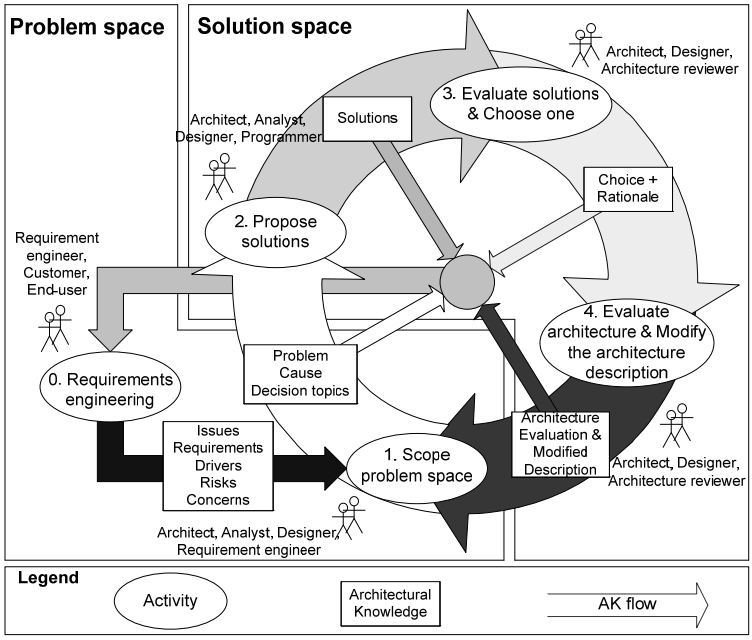

Fig. 17.1. The architecting process from an AK perspective

(0) **Requirements engineering**. This activity fuels the architecting process with different elements (e.g. requirements, drivers, decision topics, risks, and concerns) from the problem space. These form the main input for the activity of scoping the problem space. Requirements engineers, customers and end-users are typical stakeholders.

(1) **Scope problem space**. The architect selects the architecturally significant elements from the problem space and distills them into a concrete problem. To put the problem in perspective, a cause (e.g. from technical aspects) of the problem is described as well. This scoping is needed, as the problem space is usually too big, thereby forcing the architect to focus only on the key issues. Typical stakeholders of this activity are: architects, analysts, designers, and requirements engineers.

(2) **Propose solutions**. The architect uses the existing architecture description and the problem of the previous step, in order to come up with one or more solutions that (partially) address the problem. Architects, analysts, designers, and programmers are typical stakeholders in this activity.

(3) **Evaluate solutions & choose one**. The architect evaluates the solutions, and makes a design decision by selecting among the proposed

solutions (according to the evaluation results). The decisions may entail making one or more trade-offs and is accompanied by the appropriate rationale. Architects, designers, and architecture reviewers are typical stakeholders of this activity.

(4) **Evaluate architecture & modify the architecture description**. Once a solution is chosen, it is integrated in the architecture and the whole architecture is evaluated. Based on the evaluation results, the architecture description has to be modified to reflect the new status. Architects, designers, and architecture reviewers are typical stakeholders.

The collaboration activities in architecting takes place in two dimensions: horizontally and vertically. Horizontal collaboration occurs between sequential software development activities, which can be in the macro- or micro-level of the software development phases, e.g. from RE to architecting (the macro-level), or within architecting (the micro-level) from architectural analysis to architectural synthesis. In horizontal collaboration, the output, of one activity becomes the input for the subsequent activity, e.g. the output of the RE activity (i.e. a requirements specification), acts as the input of the architecting activity. On the other hand, vertical collaboration happens when different people work on the same software development activity, e.g. several designers make a class diagram using a UML tool collaboratively in the design activity [32]. In this chapter, we cover the RE and architecting activities in both collaboration dimensions. The next section elaborates on the tool suite that supports the different parts of this process, and emphasizes on the various collaboration aspects.

The proposed process is meant to be generic enough so that it can be customized and adapted into specific architecting processes used in organizations. As an example, we describe how it can be mapped to the generalized model of architecting proposed in [17]: architectural analysis maps to the scoping of the problem space (activity 1); architectural synthesis maps to proposing solutions (activity 2); architectural evaluation maps to evaluating alternative solutions and selecting the optimal one (activity 3), as well as evaluating the architecture with the integrated design decisions (activity 4). The advantage of this general applicability is that it does not conflict with established architecting processes in the organizations. The disadvantage is that it does not contain enough details to be applied on its own; it has to be refined before it can be applied in practice.

17.4 The Knowledge Architect Tool Suite

To support the collaborative architecting process described in the previous section, we implemented the Knowledge Architect (KA): a tool suite[1] for creating, using, translating, sharing, and managing AK. The process itself is described in a generic way and does not delve into details about the various aspects of collaboration, as it is meant to be as broadly applicable as possible. On the contrary, the KA tool suite entails specialized support for integrating the various process activities and supporting collaboration between the stakeholders. In specific, the tool suite implements the following features to serve the collaboration purposes:

- A central *knowledge hub*. In a large project, multiple stakeholders are involved in the different process activities and typically manage and maintain their part of the relevant AK. The knowledge hub is critical for gathering all the AK in one resource, and providing an interface to all involved stakeholders to manage and evolve it;
- *Traceability management*. In a collaborative architecting process, AK entities are produced by various stakeholders. Traceability needs to be established between these collaboratively produced artifacts (e.g. a requirement leads to a design decision and when one changes the other needs to be updated). This is of paramount importance during the architecture iterations, but also for the architecture evolution;
- *Knowledge translation among different stakeholders*. Typically stakeholders come from different backgrounds and have their own perspectives on architecture, usually limited to individual AK entities (see Fig. 17.2). Effective knowledge translation (dashed arrows in Fig. 17.2) enables various stakeholders to understand each other and speak through a "common language". Furthermore, knowledge translation provides the ability to present the "big picture", and especially the complex relationships between different parts of the knowledge;
- *Automated checking*. Different stakeholders working at varied activities and at different times may touch upon the same or related AK entities. Automated checking may help to identify the conflicts, inconsistencies, and incompleteness in the collaboratively produced AK entities. Especially, when the amount of knowledge increases, this type of automated support is the only way to effectively manage it.

[1] Part of the tool suite can be downloaded from http://search.cs.rug.nl/griffin

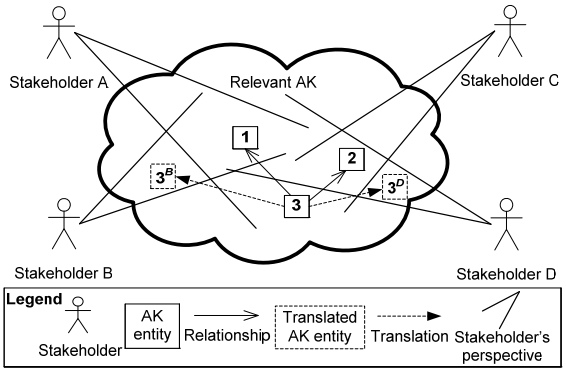

Fig. 17.2. AK sharing from the perspectives of different stakeholders

Currently, the tool suite consists of 6 tools, which are presented in Fig. 17.3: Knowledge Repository, Document Knowledge Client, Excel Plug-in, Python Plug-in, Knowledge Explorer, and Knowledge Translator. The figure illustrates how these tools are mapped onto the architecting process and its associated activities (see Fig. 17.1).

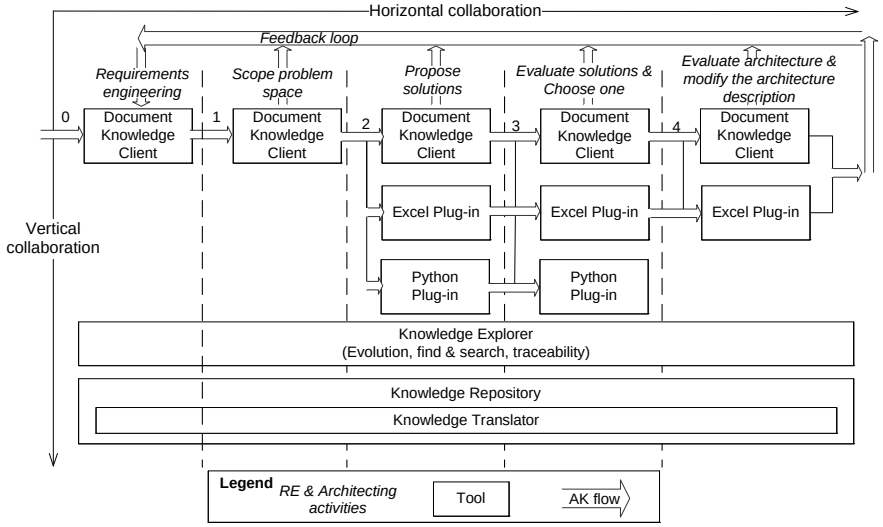

Fig. 17.3. Mapping the KA tool suit onto the requirements engineering and architecting activities

A brief outline of each tool is provided here. A more elaborate description is presented in the next subsections, while the exact details can be found in [28]. In short, these tools are the followings:

- **Knowledge Repository** is at the heart of the tool suite: a central location, which provides various interfaces for other tools to store and retrieve AK.
- **Document Knowledge Client** is a Word plug-in that supports capturing (annotating) and using (storing and retrieving from the Knowledge Repository) AK within architecture and requirement documents inside Microsoft Word.
- **Analysis Model Knowledge Clients** support capturing (annotating) and using (storing and retrieving from the Knowledge Repository) AK of quantitative analysis models. This type of analysis concerns the investigation of alternative architectural solutions by delivering (scenario-based) quantifications of one or more quality attributes of these solutions. Specifically, two knowledge clients are developed (Excel and Python Plug-in):
  - **Excel Plug-in** supports capturing and using AK of quantitative analysis models inside Microsoft Excel [20].
  - **Python Plug-in** supports capturing AK from quantitative analysis models described in Python.
- **Knowledge Explorer** analyzes the relationships between AK entities. It provides various visualizations to inspect AK entities and their relationships.
- **Knowledge Translator** (semi-)automatically translates the formal AK based on one AK domain model into the AK based on another, so that various stakeholders can understand each other when they use different AK domain models to document AK.

We have mentioned before that the KA tool suite was built in the context of the Griffin project[2] for use within our industrial partner: Astron. Therefore certain tools of the suite are aimed at integrating with the tools already used at Astron. In particular this covers Microsoft Word for architecture documentation, Microsoft Excel and Python for architecture analysis models. This is only one way to support the architecting activities (see Fig. 17.3); various other tools could be potentially built on the same underlying ideas of annotating AK on documentation and analysis models.

In this section, we first introduce these tools, including the motivations of (why) and functions provided by (what) these tools. In the next section, we present the RE and architecting activities in a collaboration perspective by using these tools in a concrete running example.

[2] GRIFFIN: a GRId For inFormatIoN about architectural knowledge, http://griffin.cs.vu.nl/

17.4.1 Knowledge Repository

The Knowledge Repository, as depicted in Fig. 17.4, is a central location for storing and retrieving AK across a wide range of architecting activities. The tool makes heavy use of technologies developed for the semantic web. For example, the open source RDF store Sesame[3] is used for storing and querying AK, while OWL (Web Ontology Language) is used for modeling AK domain models. The Knowledge Repository API provides the interfaces to communicate with all the Knowledge Clients (Document Knowledge Client, Excel and Python Plug-ins) to store the annotated AK into the repository. The Query Engine is used to query the AK entities and their relationships in the repository, and visualize them in the Knowledge Explorer. The Knowledge Translator performs the automatic translation. All the surrounding tools are described in the remaining part of this section.

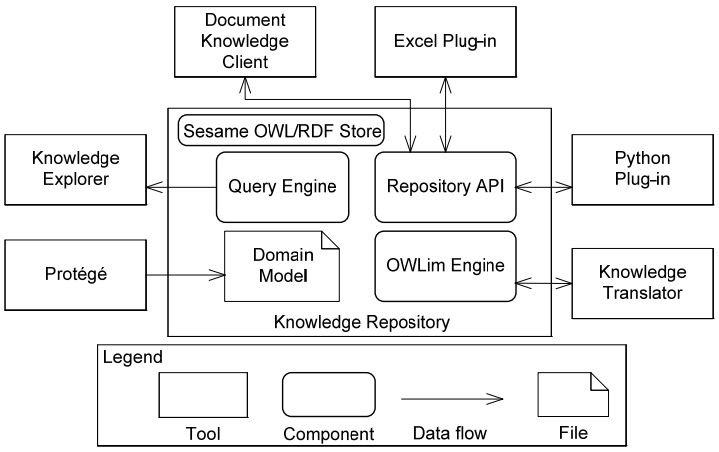

Fig. 17.4. The Knowledge Repository with other tools in the KA tool suite

17.4.2 Document Knowledge Client

The Document Knowledge Client is a plug-in to capture and use explicit AK inside Microsoft Word 2003. Various AK domain models can be deployed in the Knowledge Repository for different users (stakeholders), who annotate the AK using the AK domain models they can understand. Hence, the tool can be reused with other AK domain models. The tool offers three basic functions:

[3] http://www.openrdf.org/

AK capturing: Knowledge can be captured in a Word document by selecting a piece of text and right clicking and choosing the appropriate option from the pop-up menu. When adding a new AK entity, a menu appears, which allows the user to provide additional information about the entity, e.g. *Name*, *Type*, *Status* and *Connections*.

AK traceability: The relationships among AK entities comprise critical traceability information in collaborative architecting. For example, to find out "*who (stakeholders) are concerned with a design decision*". The AK traceability can be easily created or removed by pop-up menus in the Document Knowledge Client.

Design maturity assessment: One of the advantages of formalized (annotated) AK is automatic reasoning support based on the underlying formal models. The Document Knowledge Client supports the architect in assessing the completeness of the architecture description. Based on the AK domain model, the tool performs model checks using conformity rules to identify incomplete parts.

17.4.3 Excel Plug-in

The Excel Plug-in implements a domain model for quantitative architecture analysis models in Microsoft Excel. The tool supports analysts in making the AK produced during architecture analysis explicit. The aim is to facilitate the sharing of AK to other analysts and the analysis results in a transparent manner to other stakeholders. The tool offers the following three basic functions:

AK capturing: The major part of the AK of an architectural analysis model in Excel is found in the cells. Often labels surrounding the cell denote the semantic meaning of a cell. The tool allows analysts to make special annotations to cells. For reviewing purposes, the tool also tracks the review state of each cell and allows for comments.

AK traceability: An important feature of the tool is that it is capable of automatically inferring the dependencies among the cells (AK entities). Hence, the traceability relationships between AK entities are automatically captured.

AK visualization: To facilitate manual verification, the tool offers a visualization of the AK dependency graph, which corresponds to the cells in the Excel worksheets.

17.4.4 Python Plug-in

Similar to the Excel Plug-in, the Python Plug-in provides functionality to codify the AK of analysis models. In this case, the analysis models are expressed using the Python programming language. Both the Excel and the Python Plug-in assume quite similar domain models. Hence, the concepts and functionality discussed in the previous section also apply here.

17.4.5 Knowledge Explorer

Typically, the size of an AK repository will be considerable containing thousands of AK entities. Finding the right AK entity, or even worse a number of related AK entities, from such a big collection is not trivial. Hence, there is a need for a tool to assist in exploring an AK repository. The Knowledge Explorer can support users in visualizing AK entities and their relationships. Fig. 17.5 presents a screenshot of the tool. It provides a search functionality on the left hand side. The resulting AK entities of this search action are shown in the list on the left hand side. The results can be filtered using the drop down box on the left, thereby reducing the size of the found results. The filtering is based on the type of the AK. The available options are taken from the used AK domain model. Double clicking on one of the search items results in illustrating a number of related AK entities in columns.

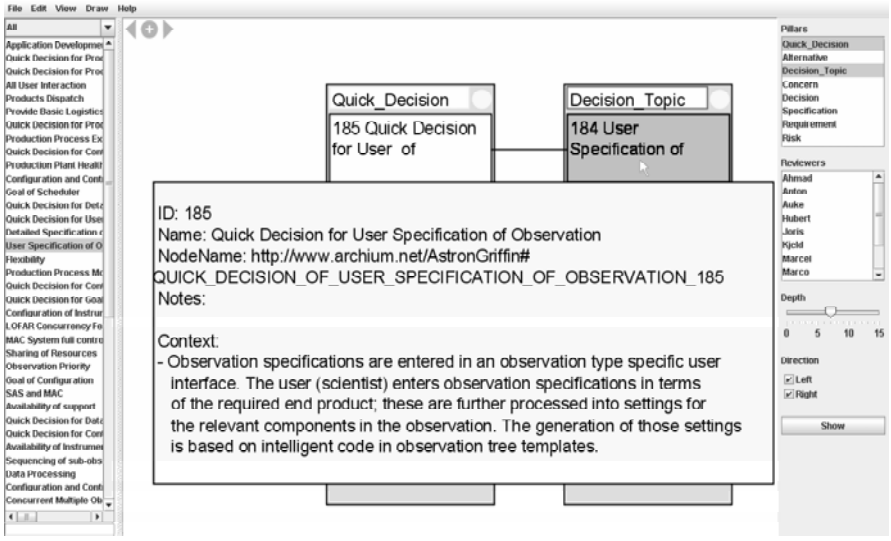

Fig. 17.5. The screenshot of the Knowledge Explorer

17.4.6 Knowledge Translator

The purpose of the Knowledge Translator is to translate the AK in various AK domain models from one to the other and vice versa. This allows various users to understand the AK codified in different AK domain models. This is critical for stakeholders from different backgrounds to understand each other in a collaborative architecting process. For example, a requirements engineer and an architect use different AK domain models to produce and consume requirements (part of AK), but need to have a common understanding. Currently, we employ the core model proposed in [10] as a central model for the AK translation by an indirect translation approach [27].

The AK translation can be done manually or automatically. Both ways have their respective advantages and disadvantages on translation cost and quality, and stakeholders can select an appropriate manner by trading off quality and cost in their own context. The initial cost-benefit analysis about the AK translation cost and quality has been investigated in [27].

17.5 Collaboration within the Process with KA

In this section, we present the collaboration within the proposed architecting process, as it is supported by the KA tool suite. We discuss both horizontal and vertical collaboration and demonstrate them through a running example. The context of this running example originates from the architecting process used at our industrial partner, Astron (see Sect. 17.3). In their projects, there is a large and complex body of knowledge that needs to be shared frequently among the distributed stakeholders. However, the different backgrounds and expertise of these stakeholders restrains them from achieving a common understanding and thus hinders the integration of collaborative architecting activities. We have worked closely with Astron for the software architecture of two projects that concern the next generation of radio telescopes. The stakeholders involved with the architecting process in these projects include end-users (scientists), requirements engineers, architects, analysts, designers and architecture reviewers.

17.5.1 Requirements Engineering

Horizontal Collaboration

In a traditional software development scenario, a requirements engineer produces the software requirements specification in a document, e.g. in a

Word file. The requirements engineer subsequently delivers the requirements documentation to the architect for the architecture design. Within this process, the requirements engineer, architect, and other related stakeholders will closely interact with each other. This close interaction is needed to ensure common document understanding [6], conciliate requirements [31], and improve the architecture design, etc. In a distributed development environment or in a long-term development project, this intensive interaction between the requirements engineer and the architect is quite challenging. The geographical distance between the two actors hinders effective interaction, while staff reassignment in a long-term project would result in knowledge vaporization [19]. In such cases, the Knowledge Repository acts as the project requirements knowledge center: the repository provides valuable requirements information according to established AK domain models[4], and it helps the architect to understand the requirements correctly and unambiguously.

Running example: a requirements engineer[5] specifies the requirements (including *architectural significant requirements*, *concerns* and *risks*, etc.) in the requirements document through discussion with customers. Afterwards, the requirements engineer uses the Document Knowledge Client to annotate the knowledge about requirements in this document, e.g. "*The user (scientist) uses these interfaces to propose and specify observations.*" (an AK entity of concept *Requirements*), and "*This flexibility is of great importance especially for the high performance applications.*" (an AK entity of concept *Concerns*). In the end, all the annotated AK entities are stored into the Knowledge Repository. The architect retrieves the requirements information from the Knowledge Repository, and scopes the problem (architectural analysis) by choosing only the architecturally-significant ones (e.g. scoping the *decision topics* from the *requirements*). The architect subsequently stores the newly produced AK entities into the Knowledge Repository for further collaboration.

The whole collaboration process is illustrated in Fig. 17.6. The numbers in this figure represent the actions sequence. The KA tool suite offers features to support these collaboration activities. For example, the design maturity assessment function based on formal AK can help the architect to

[4] If there is no explicit specification, we assume that the AK domain model employed in various requirements engineering and architecting activities for producing and consuming AK is the same one, so that all stakeholders can communicate the AK in a common language.

[5] The collaboration between other stakeholders is also critical, e.g. between the telescope user and requirements engineer, but we focus on the requirements engineer and architect in the scope of this chapter.

find out whether all the *requirements* have been considered or not. Another example is that the traceability of formal AK can help the architect to trace from the design space (e.g. a *design decision*) back to the original cause in the problem space (e.g. a *requirement*).

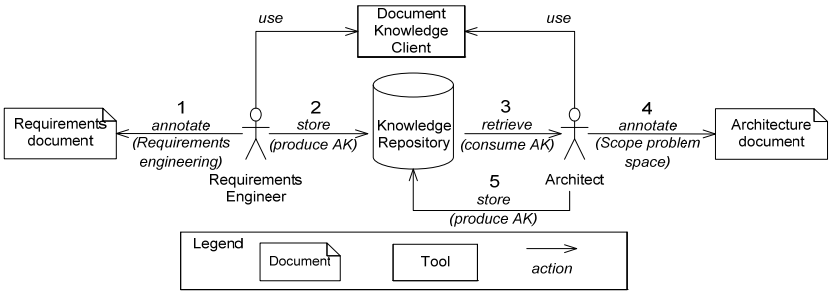

Fig. 17.6. AK sharing process between requirements engineer and architect

Vertical Collaboration

The typical scenario in RE is that all the system stakeholders can propose their individual *requirements*, *concerns*, and *risks* from different perspective and at different levels (business goals, product features, user requirements, etc.). Inevitably, there are always conflicts (e.g. conflict business goals, concerns) and mismatch (e.g. no user requirements relating to a product feature) in the candidate requirements. The collaboration among all the requirements stakeholders is needed to form a clear and unambiguous requirements specification using negotiation and reaching compromises. Another situation is that different requirements engineers work on the requirements specification for different part of the system at same time. In this case, they also have to understand the requirements, which have been elicited and documented by other requirements engineers for consistency. Hence, collaboration among these requirements engineers is a necessity to achieve a coherent and consistent requirements specification.

Running example: Customer *A* specifies the requirement "*The flow of information, either control or monitoring metrics, is in the vertical direction.*", and then the requirements engineer uses the Document Knowledge Client to annotate this *requirement* and store this AK entity into the Knowledge Repository. Customer *B* uses the Document Knowledge Client to retrieve the latest *requirements* from the Knowledge Repository. After this, Customer *B* finds out that the *requirement* "*The flow of information in the vertical direction*" is not desirable. The customer wants "*The flow of information is in the horizontal direction*". In this situation, Customer *B* adds his/her requirement, annotates, and stores this requirement as a con-

flict requirement with the requirement proposed by Customer *A*. Eventually, the requirements engineers will try to negotiate and resolve the conflict with all the other requirements stakeholders (e.g. through voting) or just inquire the high level project decision maker to choose one.

17.5.2 Scope Problem Space

Horizontal Collaboration

"Scope problem space" is the first activity in the architecting process, aimed at refining the problem space by selecting the architecture significant problem elements. The results of this activity are a set of *architectural significant requirements*, e.g. *problem*, *cause*, and *decision topic*s, which are further used in the following activity to produce *alternative architectural solutions*. The architect uses the Document Knowledge Client to annotate these *architectural significant requirements*, which he/she has identified, using the AK domain model, and stores them into the Knowledge Repository. After this, the analyst can retrieve this AK from the Knowledge Repository, understand it based on the AK domain model, and propose *alternative architectural solutions.*

Running example: An architect analyzes an *architectural significant requirement*, e.g. "*In this (data) view on the system software, we focus on the control over the data processing pipelines.*", and gets a *decision topic*, e.g. "*the control method over the data processing pipelines*", which has to be addressed by a *design decision.* After that, the architect annotates and stores this *decision topic* into the Knowledge Repository. The *decision topics* can be retrieved by the analyst from the Knowledge Repository for further collaboration, e.g. in the proposing solutions activity.

17.5.3 Propose Solutions

Horizontal Collaboration

Once the scoping of the problem space is complete and a clearer picture of the problem at hand is created, the architect has to define one or more alternative solutions to (partially) address the problem. These alternatives need to be shared in some shape or form, e.g. using a textual description, figures, presentation, or a conversation, in order to be evaluated. For important decisions, the alternatives are shared with the stakeholders: (1) to validate whether the alternative is indeed addressing the problem (2) to create understanding and support among the stakeholders for the choice made in the next step.

Furthermore, thinking up alternative solutions often leads to new insights in the problem space. For example, it is not uncommon to find requirements unclear on key aspects or find out that a particular concern is being overlooked. Hence, close collaboration with a requirements engineer (and perhaps other stakeholders as well) is needed to sort out these aspects.

Running example: Following the running example from the previous activity, the analyst retrieves this *decision topic* from the Knowledge Repository, and proposes several *alternative architectural solutions*, e.g. "*use real-time control method*", "*use batch control*" and "*use real-time or batch control depending on the data characteristics*" After this, the analyst annotates these *alternative architectural solutions* in the architecture document and stores these newly produced AK entities into the Knowledge Repository. The architecture reviewer retrieves the corresponding *concerns*, *decision topic*, and its *alternative architectural solutions* from the Knowledge Repository. Based on this AK, the reviewer evaluates the *alternative architectural solutions* against related user *concerns*. It is noted that there is a bidirectional traceability relationship created automatically between a *decision topic* and an *alternative architectural solution*, as dictated by the relationships in the AK domain model. With the bidirectional traceability relationship, when the architect changes (removes, modifies) the *decision topic*, then the analyst will be notified to reconsider the *alternative architectural solutions* which have been proposed.

Vertical Collaboration

For two reasons the proposed alternatives need to be shared among architects as well. Firstly, sharing alternatives among each other inspires architects to consider new solution directions. Often this takes the form of creatively combining existing alternatives into a new one. Secondly, this sharing prevents architects from redoing work already done by their peers. For analysts, sharing the alternatives is important as well. The analysis of different experts has to be reconciled to evaluate a single alternative. However, this requires a shared understanding among the analysts what this alternative exactly entails. Consequently, the knowledge of what these alternatives are should be shared.

Running example: The analysts use the Knowledge Explorer to find out what kind of assumptions their fellow analysts have made in their analysis about the alternatives. Based on this knowledge, they can update their own analysis models. Software architects can share a software architecture document to facilitate vertical collaboration. Using the Document Knowledge Client, an architect can trace from a *Decision Topic* to the proposed alternatives and read their description.

17.5.4 Evaluate Solutions & Choose One

Horizontal Collaboration

The horizontal collaboration in this activity takes place between the software architect/analyst and other stakeholders. It involves sharing four different types of AK. The first type is the evaluation criteria that should be used to judge the various alternative solutions. An important criterion is the extension to which a proposed alternative solution addresses the defined decision topic. In addition, the captured concerns during RE provides good candidates for evaluation criteria. Additional horizontal collaboration with the requirements engineers is needed when the evaluation criteria are not clear.

The second type is the relative importance of the aforementioned criteria. Typically, there are differences among how the stakeholders perceive the importance of the criteria. Hence, the architect has to reach an acceptable compromise, and through horizontal collaboration, communicate this compromise to the stakeholders.

The third type is the perceived pros and cons of each alternative, i.e. the ranking of each proposed alternative solution on the defined criteria. Often conflicts arise among stakeholders due to differences in the perception of these pros and cons and their associated likelihood and strength. Since this knowledge forms the basis of the rationale of the choice, it is of paramount concern to reach consensus among the stakeholders about these properties. One of the goals of analysts is providing detailed information about these properties in an objective manner to facilitate this ranking.

The fourth type is the choice made among the alternatives. The associated rationale is based on the three earlier introduced elements. In practice, only this last element is typically communicated. In this situation, the rationale and the three other elements are only shared when asked for.

Running example: In the previous step, three *alternative architectural solutions* were proposed and documented in a document: "*use real-time control method*", "*use batch control*" and "*use real-time or batch control depending on the data characteristics*". In this step, the architect writes down the choice made (e.g. for the use real-time control method) and provides a small explanation for this choice, e.g. reducing costs by not requiring additional storage. Selecting this piece of text and pressing the add KE button of the Document Knowledge Client adds the text as a *Decision* to the Knowledge Repository. To provide traceability, the architect relates the newly created *Decision* KE to the chosen *Alternative*. Indirectly, this also relates the *Decision* to the other considered alternatives through their common *Decision Topic*.

To provide rationale, the tool suite provides two options. The first one is found in the Document Knowledge Client and allows the architect to relate an analysis result from one of the Analysis Model Clients (Excel and Python Plug-ins) as either a *Pro* or *Con* to an *Alternative*. For example, the predicted cost of the real-time alternative. The second option is to use the Knowledge Explorer to find suitable concerns (e.g. cost) that could be an evaluation criterion.

Vertical Collaboration

Among analysts the vertical collaboration for this activity mostly consists of unifying the analysis results of different experts in one consistent picture. In this way, evaluating the alternatives becomes relatively easy. Vertical collaboration among architects is about the knowledge sharing covering the aforementioned four AK types, since it is this knowledge that makes up the reasoning behind the architecture.

Running example: To present an objective basis for decision making analysts make a four column table in the architecture document with the first column being the criteria used and the other three columns representing the three alternatives considered. The rows present for each criterion the analysis result for each alternative. Using the Document Knowledge Client, the analyst creates the traceability between the document and his/her quantitative analysis from Python or Excel. By sharing this document with other analysts, each adding their own row, a complete unified picture for the evaluation is created. Architects use a similar approach.

17.5.5 Evaluate Architecture & Modify the Architecture Description

This evaluation activity is similar to the previous evaluation activity, but has a larger scope. The previous activity focuses on the evaluation of alternative architecture solutions while this activity evaluates the entire architecture with the incorporated new design decision (chosen solution). Consequently, the collaborations through AK sharing of these two evaluation activities are quite similar. Hence, we do not repeat them again. We focus on the activity “modify the architecture description”.

Horizontal Collaboration

Collaboration in this activity happens between sequential activities, i.e. horizontal collaboration from architecture description to detailed design. In this collaboration, the Knowledge Repository can also act as the hub in

which the architects and designers share the architecture description information.

Running example: An architect makes a *design decision* "*use real time control during data taking and processing*", annotates, and stores this AK entity into the Knowledge Repository. A designer retrieves the latest *design decisions* from the Knowledge Repository and makes a detailed design which is based on this *design decision*.

Vertical Collaboration

Based on the evaluation results, an architect modifies the design and documents the outcome of design, using natural language or special notations (e.g. Architectural Description Language or UML) in a document. The architecture description can be completed by a single architect in a small project, but for a large project, several architects will be working together for the various parts of the system. The collaboration among them is essential to produce an integrated and consistent architecture document in the end. The Knowledge Repository acts as the hub in which all the architects share the architecture description information with each other.

Running example: One of the user *concerns* about the system is stated as "*Performance issue is in a higher priority than cost in this system*". Architect *A* makes a *design decision* to address this *concern* as "*use real time control during data taking and processing*", and annotates and stores this AK entity into the Knowledge Repository. Architect *B* makes another *design decision* to address the same *concern* as "*limit the data payload during data taking and processing*", annotates, and stores this AK entity into the Knowledge Repository as well. Architect *C* retrieves the latest *design decisions* from the Knowledge Repository and uses the design maturity assessment function provided by Document Knowledge Client to verify the architecture design. The design maturity assessment function detects that these two *design decisions* address the same *concern* and are actually in conflict with each other. Therefore, Architect *C* tries to negotiate with Architects *A* and *B* to come up with a single *design decision*, e.g. "*use real time control during data taking and processing*". Other defects or weak points can also be detected by the design maturity assessment, such as incompleteness. Architect *C* annotates the new *design decision* and stores (updates) the Knowledge Repository for further collaboration with other architects.

17.5.6 Feedback Loop

Feedback can be provided from any architecting activity to the RE activity, as for example new user *concerns*, *solutions* and *design decisions* pose new *requirements*. Architecting is a highly iterative process. In each iteration, the requirements are revisited until all the *architectural significant requirements* are satisfied and all *risks* are mitigated. The Knowledge Repository is the central storage of AK produced in all activities, and supports feeding this knowledge back to the RE activity.

Running example: An example of collaboration that concerns providing feedback to RE is the following: the architect makes a *design decision* "*use SAS (a software package for data visualization) for data observation*", annotates, and stores this *design decision* into the Knowledge Repository. A requirements engineer retrieves this *design decision* from the Knowledge Repository and finds that this *design decision* results in a new *requirement* "*the data observation should be visualized in GUI*". The requirements engineer annotates and stores this newly-produced requirement into the Knowledge Repository. In this way, (other) requirements engineers can retrieve the updated requirements from the Knowledge Repository and validate the consistency between the new requirement and the existing ones.

17.5.7 Architectural Knowledge Translation

AK translation is a common function in all activities (both RE and architecting), since the involved stakeholders typically use different AK domain models to produce and consume the AK. It is comparable to human language translation, were people from different countries speaking different languages try to communicate. A translator is needed for effective communication between them, as he or she translates from one language to another and vice versa. The quality of the translation depends on the quality of the translator, i.e. how correctly the translator can translate knowledge. In AK translation, various translation methods can be employed with their specific advantages and disadvantages depending on the translation context (number of involved AK domain models and AK entities, etc.) [27][26].

Running example: A requirements engineer working at branch *A* of Astron uses the AREL AK domain model [25] to annotate knowledge about *requirements* e.g. "*The user (scientist) uses these interfaces to propose and specify observations*" (an AK entity of AREL concept *Functional requirement*), and "*The new user (scientist) shall know how to use these in-*

terfaces to propose and specify observations in 2 hours" (an AK entity of AREL concept *Non-functional requirement*). These two AK entities are subsequently stored into the Knowledge Repository. An architect working at branch *B* of Astron uses the LOFAR AK domain model [18] to consume and produce AK. In particular, the architect uses the concept *Requirement* from the LOFAR AK domain model to retrieve all the requirements information from the Knowledge Repository which has been produced by the requirements engineer of branch *A*. Due to the different requirement concepts being used by the AK producer (requirements engineer at branch *A*) and consumer (architect at branch *B*), knowledge translation is needed. The Knowledge Translator uses the defined AK concept mapping relationship to translate AK entities. For example, the AREL AK concept *Functional requirement* and *Non-functional requirement* are both the *subClassOf* the LOFAR AK concept *Requirement*. Using this relationship, the Knowledge Translator translates the two AK entities annotated in the AREL domain model into the AK entities in the LOFAR domain model and stores translated AK entities into the Knowledge Repository. After this translation, the architect at branch *B* can retrieve all the requirements information from the Knowledge Repository.

17.6 Related Work

Computer Supported Cooperative Work (CSCW) in software engineering comprises all software engineering methods, norms, and tools that support teamwork flexibly and effectively [7]. CSCW concentrates on improving the efficiency of groupware [25] for software development. It focuses on the vertical collaboration in the software development lifecycle, e.g. the collaboration among requirements engineers or among designers. One such example is ProjectIT-Studio, an integrated environment that supports collaborative RE by combining wikis with CASE tools for requirements specification and validation [14]. This tool can assist non-technical stakeholders during the requirements specification and help requirements engineers for a seamless integration with dedicated RE CASE tools. ProjectIT-Studio fosters the stakeholders' involvement in collaborative RE from a socio-technical perspective. Another example is the UML profile UML-G for cooperative UML modeling in the design activity [32]. It supports software modeling by explicitly representing shared data, roles and actors in cooperative sessions. UML-G stresses the sharing of design outcomes (i.e. models), but does not pay attention to the rationale underneath the design.

A CSCW approach for architecting was proposed in [15], addressing the collaborative architecture modeling of complex component-based systems. A collaborative modeling tool was provided for the architecture design team in which several architects design an architecture cooperatively. Multiple architects are able to concurrently access and manipulate the software architecture information stored in a server machine. The shared software architecture information in this tool is mostly the design artifacts (e.g. components, data flows, external entities, etc). There is no support to store information about design decisions and rationale.

Similarly, Maheshwari and Teoh implemented a web-based tool for collaborative software architecture evaluation, supporting the Architecture Tradeoff Analysis Method (ATAM) [29]. They argue that the ATAM method has its limitations in an increasingly globalized software industry in which the distribution of development teams is extensive. Their web-based tool provides a mental mapping from the physical world to the internet world. For example, their tool set provides communication tools, such as chatting, brainstorming, voting tool, etc. The tool set also provides some assistant tools for ATAM, such as Utility Tree Viewer/Editor, Features Evaluator, etc. Most of the knowledge exchanged by their tool set is personalized knowledge, which is often difficult to understand by users who come from different backgrounds.

Farenhorst et al. use wikis to support collaboration, communication, and consensus decision making in the architecting process of distributed development by sharing AK [13]. They suggested that, for successful AK sharing, it is necessary to tailor the types and content of AK for sharing according to the concrete architecting process [12]. Their work focuses on personalized (e.g. by using yellow pages) and documented AK and not on formal AK.

PAKME (Process-centric Architectural Knowledge Management Environment) is a web-based tool aimed at providing knowledge management support for the architecting process [1]. PAKME focuses on various collaborative features (e.g. collaborative decision making) for distributed stakeholders involved in the architecting process by managing codified AK (pattern, decision etc.) and personalized AK (contact management, online collaboration, etc). Other related work on AK sharing and reusing can be found in the SHARK workshop series [1][3][5].

17.7 Conclusions and Future Work

AK is widely accepted and recognized to be of paramount importance for the success of software architecting. However, the collaboration among the stakeholders involved in the architecting process is hindered by the lack of integration of architecting activities and the corresponding AK. This has severe implications for the quality of both the architecting process and the product. This chapter presented a collaborative architecting process and the accompanying tool suite, that integrate the architecting activities through AK sharing.

The process and the accompanying tool suite address the four goals of collaboration in software architecting identified in Sect. 17.2.1:

(1) using the central Knowledge Repository and Knowledge Client tools, an integrated and consistent architecture document can be produced through stakeholders collaboration;
(2) using various AK domain models to capture (annotate) AK in the Knowledge Clients, dependencies and especially traceability among architecture artifacts can be effectively managed in the Knowledge Repository;
(3) using the functions provided by the Knowledge Client tools (e.g. design maturity assessment of the Document Knowledge Client), the architectural conflicts, risks, inconsistency and incompleteness can be identified, recorded and resolved based on the formal relationships defined in the AK domain model and semantic web inferencer;
(4) using the central Knowledge Repository, all the knowledge which is relevant to the whole architecting process (AK) is recorded.

Although the proposed approach (process and tool suite) was derived from a specific organization, it is generally applicable to other organizations: as explained in Sect. 17.3, the proposed collaborative architecting process is orthogonal to current architecting processes. Due to its generic nature, it has to be adapted and customized into an existing architecting process before it is put into practice. For the accompanying tool suite, some general tools (Knowledge Repository, Document Knowledge Client, Knowledge Explorer, and Knowledge Translator) can be adjusted and employed to the architecting processes mentioned above since they follow closely the proposed process. The Excel and Python Plug-ins have been developed according to Astron's needs, and can only be used if other organizations have similar needs (quantitative analysis).

The KA tool suite has been used and (empirically) validated in two industrial case studies at Astron for quantitative analysis of architecture design [20] and enrichment of architecture documentation [18]. In [20], the

tool suite was deemed effective for facilitating AK sharing for verification and validation of quantitative architectural solutions. In [18], we proved that the tool suite helps to partially address the shortcomings of current architecture documentation approaches of large and complex systems.

In the future, the integrated collaborative architecting process with the tool suite should be further validated in a larger industrial project with a cost-benefit analysis. The tool suite needs to be further improved with respect to its usability and scalability. Finally, we plan to extend this suite with other tools for a wider application of AK sharing (e.g. UML/ADL modelers, Email Plug-in, and other quantitative analysis tools).

Acknowledgments. 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. The authors would like to thank Astron for their support and access to the LOFAR software architecture documents.

References

[1] Ali-Babar M, Gorton I (2007) A tool for managing software architecture knowledge. In: Proceedings of the 2nd Workshop on SHAring and Reusing architectural Knowl-edge - Architecture, rationale, and Design Intent (SHARK/ADI 2007), May 20-26, pp. 11-17

[2] Ali-Babar M, Gorton I, Kitchenham B (2006) A framework for supporting architecture knowledge and rationale management. Rationale Management in Software Engineering, Dutoit AH, et al., Editors, pp. 237-254

[3] Avgeriou P, Lago P, Kruchten P (2008) Third international workshop on sharing and reusing architectural knowledge (SHARK 2008). ICSE Companion, pp. 1065-1066

[4] Avgeriou P, Kruchten P, Lago P, Grisham P, Perry D (2007) Architectural knowledge and rationale: issues, trends, challenges. ACM SIGSOFT Software Engineering Notes 32(4): 41-46

[5] Bass L, Clements P, Kazman R (2003) Software architecture in practice (2nd edition). Addison-Wesley Professional

[6] Bhat JM, Gupta M, Murthy SN (2006) Overcoming requirements engineering challenges: lessons from offshore outsourcing. IEEE Software 23(5): 38-44

[7] Bischofberger WR, Kofler T, and Mätzel KU, Schäffer B (2002) Computer supported cooperative software engineering with beyond-sniff. In: Proceedings of the 7th Conference on Software Engineering Environments (SEE 1995), April 5-7, pp. 135-143

[8] Capilla R, Nava F, Carrillo C (2008) Effort estimation in capturing architectural knowledge. In: Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), September 15-19, pp. 208-217
[9] Capilla R, Nava F, Pérez S, Dueñas J (2006) A web-based tool for managing architectural design decisions. ACM SIGSOFT Software Engineering Notes 31(5): 20-27
[10] de Boer RC, Farenhorst R, Lago P, van Vliet H, Clerc V, Jansen A (2007) Architectural knowledge: getting to the core. In: Proceedings of the 3rd International Conference on the Quality of Software Architectures (QoSA 2007), July 12-13, pp. 197-214
[11] Dingsøyr T, Conradi R (2002) A survey of case studies of the use of knowledge management in software engineering. International Journal of Software Engineering and Knowledge Engineering 12(4): 391-414
[12] Farenhorst R (2006) Tailoring knowledge sharing to the architecting process. ACM SIGSOFT Software Engineering Notes 31(5): 15-19
[13] Farenhorst R, van Vliet H (2008) Experiences with a wiki to support architectural knowledge sharing. In: Proceedings of the 3rd Workshop on Wikis For Software En-gineering (Wikis4SE 2008), September 8-10
[14] Ferreira D, da Silva AR (2008) Wiki supported collaborative requirements engineering. In: Proceedings of the 3rd Workshop on Wikis for Software Engineering (Wikis4SE 2008), September 8-10
[15] Guo J, Liao Y, Parviz B (2006) A collaboration-oriented software architecture model-ing system - JArchiDesigner. In: Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS 2006), March 27-30, pp. 481-482
[16] Hansen MT, Nohria N, Tierney T (1999) What's your strategy for managing knowledge? Havard Business Review 77(2): 106-116
[17] Hofmeister C, Kruchten P, Nord RL, Obbink H, Ran A, America P (2005) A general model of software architecture design derived from five industrial approaches. Journal of Systems and Software 80(1): 106-126
[18] Jansen A, Avgeriou P, van der Ven JS (2009) Enriching software architecture documentation. Journal of Systems and Software (accepted)
[19] Jansen A, Bosch J (2005) Software architecture as a set of architectural design decisions. In: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA 2005), November 6-10, pp. 109-120
[20] Jansen A, de Vries T, Avgeriou P, van Veelen M (2008) Sharing the architectural knowledge of quantitative analysis. In: Proceedings of the 4th International Conference on the Quality of Software Architectures (QoSA 2008), October 14-17, pp. 220-234
[21] Jansen A, van der Ven J, Avgeriou P (2007) Tool support for architectural decisions. In: Proceedings of the 6th Working IEEE/IFIP Conference on Software Architecture (WICSA 2007), January 6-9, pp. 44-53
[22] Kruchten P (2004) An ontology of architectural design decisions in software intensive systems. In: Proceedings of the 2nd Groningen Workshop on Software Variability Management (SVM 2004), December 2-3, pp. 54-61

[23] Kruchten P, Lago P, van Vliet H (2006) Building up and reasoning about architectural knowledge. In: Proceedings of the 2nd International Conference on the Quality of Software Architectures (QoSA 2006), June 27-29, pp. 43-58
[24] Lago P, Avgeriou P (2006) First workshop on sharing and reusing architectural knowledge. ACM SIGSOFT Software Engineering Notes 31(5): 32-36
[25] Li J, Li T, Lin Z, Mathur AP, Kanoun K (2004) Computer supported cooperative work in software engineering. In: Proceedings of the 28th International Computer Software and Applications Conference (COMPSAC 2004), September 27-30, pp. 328-328
[26] Liang P, Jansen A, Avgeriou P (2008) Selecting a high-quality central model for sharing architectural knowledge. In: Proceedings of the 8th International Conference on Quality Software (QSIC 2008), August 12-13, pp. 357-365
[27] Liang P, Jansen A, Avgeriou P (2009) Sharing architecture knowledge through models: quality and cost. The Knowledge Engineering Review (in press)
[28] Liang P, Jansen A, Avgeriou P (2009) Knowledge architect: a tool suite for managing software architecture knowledge. Technical Report RUG-SEARCH-09-L01, University of Groningen. http://www.cs.rug.nl/~liangp/download/liang2009kat.pdf
[29] Maheshwari P, Teoh A (2005) Supporting ATAM with a collaborative web-based software architecture evaluation tool. Science of Computer Programming 57(1): 109-128
[30] Nonaka I, Takeuchi H (1995) The Knowledge-creating company: how Japanese companies create the dynamics of innovation. Oxford University Press, USA
[31] Nuseibeh B (2001) Weaving together requirements and architectures. IEEE Computer 34(3): 115-117
[32] Rubart J, Dawabi P (2004) Shared data modeling with UML-G. International Journal of Computer Applications in Technology 19(3): 231-243
[33] Rus I, Lindvall M (2002) Knowledge management in software engineering. IEEE Software 19(3): 26-38
[34] Tang A, Ali-Babar M, Gorton I, Han J (2006) A survey of architecture design rationale. Journal of Systems and Software 79(12): 1792-1804
[35] Tang A, Jin Y, Han J (2007) A rationale-based architecture model for design traceability and reasoning. Journal of Systems and Software 80(6): 918-934
[36] Tyree J, Akerman A (2005) Architecture decisions: demystifying architecture. IEEE Software 22(2): 19-27
[37] van der Ven J, Jansen A, Avgeriou P, Hammer D (2006) Using architectural decisions. In: Short Papers of the 2nd International Conference on the Quality of Software Architectures (QoSA 2006), July 27-29
[38] Whitehead J (2007) Collaboration in software engineering: a roadmap. In: Proceedings of Future of Software Engineering (FOSE 2007), March 20-22, pp. 214-225