10 The GRIFFIN Collaborative Virtual Community for Architectural Knowledge Management

P. Lago, R. Farenhorst, P. Avgeriou, R.C. de Boer, V. Clerc, A. Jansen, and H. Van Vliet

Abstract: Modern software architecting increasingly often takes place in geographically distributed contexts involving teams of professionals and customers with different backgrounds and roles. So far, attention and effort have been mainly dedicated to individuals sharing already formalized knowledge and less to social, informal collaboration. Furthermore, in Web 2.0 contexts, little to no attention has been given to practitioners carrying out complex, collaborative, and knowledge-intensive tasks in organizational contexts.

This chapter shows how we can effectively support the combination of formal and informal collaboration and build a Virtual Community for architectural knowledge sharing. We present a set of collaboration scenarios that define a conceptual model for such a Virtual Community. A solution in this area would realize the expectations of companies involved in IT and working in distributed settings to effectively exploit their expertise, and turn their professional knowledge into a global IT portfolio.

Keywords: Architectural Knowledge, Design Decisions, Knowledge Communities, Web 2.0.

10.1 Introduction

The notion of software architecture is one of the key technical advances to the field of software engineering over the last decade. The advantages of using explicit software architecture include early interaction with stakeholders, its basis for establishing work breakdown structure and early assessment of quality attributes [1].

The GRIFFIN project develops notations, tools and associated methods to extract, represent, and use architectural knowledge that currently is not documented or represented in the system. In GRIFFIN, Architectural

Knowledge (AK) is defined as the integrated representation of the software architecture of a software-intensive system or a family of systems, the architectural design decisions, and the external context/environment. The project emphasizes sharing architectural knowledge in a distributed, global context. Some of the results can be found in [2, 9].

GRIFFIN is a joint research project of the VU University Amsterdam and the University of Groningen, both in the Netherlands. The research is carried out in a consortium with various industrial partners, both large and small. These partners provide us with case studies and give feedback. The domains of these case studies range from a family of consumer electronics products to a highly distributed system that collects scientific data from around 15,000 sensors to a service-oriented system in a business domain.

Although considerable progress has been made, we still lack techniques for capturing, representing, and maintaining knowledge about software architectures. While much attention has been given to documenting architectural solutions, the rationale for these solutions often remains implicit and is often exchanged in interpersonal, informal communication. The incomplete representation of the needed AK leads to several problems that are generally recognized in any software engineering project, and that become just worse in distributed and global software development:

- **Lack of first-class representation** [10] architectural solutions, design decisions, and rationale lack a first class representation in the software architecture. Consequently, the knowledge about the "what and how" of the software architecture is quickly lost. Experience shows that this documentation on architecture design decisions is difficult to interpret and use by individuals not involved in the initial design of the system.
- **Architectural knowledge is cross-cutting and intertwined** [10] architectural knowledge addresses technical, business, organizational, and cultural aspects that influence architectural decisions and design solutions. Due to its inter-disciplinary nature, architectural knowledge is cross-cutting, affecting multiple components and connectors, and one piece of architectural knowledge often becomes intimately intertwined with another piece of architectural knowledge.
- **High cost of change** [10] a resulting problem is that a software architecture, once implemented, is prohibitively expensive to change. Moreover, changing or removing existing design decisions is difficult.

- **Design rules, constraints, and rationale violated** [10] during the evolution of software systems, designers and even architects may easily violate the design rules, constraints, and rationale imposed on the architecture during earlier design iterations.
- **Obsolete design decisions not removed** [10] removing obsolete architecture design decisions from an implemented architecture is typically avoided, or performed only partially, because of (1) the effort required, (2) perceived lack of benefit and (3) concerns about the consequences, due to the lack of knowledge about them. The consequence is a rapid erosion of the software system, resulting in high maintenance cost.
- **Architectural knowledge is dispersed and undocumented**: documented or formalized architectural knowledge is usually limited to technical architectural solutions. Non-technical knowledge such as business and cultural aspects remains tacit and only known to individuals. This architectural knowledge is then lost, and difficult (if not impossible) to trace back and reuse in later developments.
- **Documented architectural knowledge neglects interdisciplinary use**: architectural knowledge documentation should convey the overall architecture to persons with different culture, skills, and responsibilities in different architectural aspects or subsystems. Persons working at the subsystem level easily lose track of relations between their "part" and the overall architecture. This hampers traceability and may lead to changes that conflict with the general architectural decisions, which instead should orchestrate the differences in the involved parties.

When software engineering projects are distributed or global, the problems above are aggravated. Knowledge transfer is a communication process requiring strict interaction and agile information exchange. In local software development, it is already difficult to rationalize the type and amount of knowledge we need to exchange. If in addition exchanges occur remotely and via a technological infrastructure, we have to make this knowledge explicit, and we need to identify agile means to render this process as dynamic and powerful as possible.

In this chapter, we describe the conceptual collaborative scenarios implementing a virtual community aimed at sharing architectural knowledge in a distributed setting. As envisaged by Zhuge [11] "Modern communication facilities like the Internet provide people with unprecedented social opportunities for knowledge generation and sharing". To improve this knowledge generation and sharing, Zhuge designed a knowledge grid that supports social activities in different environmental spaces. In our work we aim at realizing such a knowledge grid for professionals involved in the software architecture processes. To this end, we first highlight some trends in architectural knowledge representation and sharing. Then, we define the

collaboration requirements for the GRIFFIN virtual community followed by the scenarios realizing them. We further show how this set of scenarios combine formalized and informal AK sharing; a combination that can be finally mapped on Web 2.0 services.

10.1.1 From a codification/personalization to a hybrid knowledge management strategy

In most literature, e.g., [12] knowledge is classified into tacit, documented, and formalized knowledge. *Tacit knowledge* (e.g., organization strategies or best practices) is implicitly known and used by software architects, but not made explicit. *Documented knowledge* about software architecture (e.g., design decisions or rationale) can be interpreted and used by humans, whereas *formalized knowledge* (e.g., domain-specific ontologies) can be created and used by both humans and software systems.

In software development organizations much knowledge is kept in unstructured forms: FAQs, mailing lists, email repositories, bug reports, lists of open issues, etc., Lightweight tools such as wikis, weblogs, and yellow pages are other examples of relatively unstructured repositories to share information in global projects.

In the knowledge management literature, a distinction is made between a *personalization strategy* and a *codification strategy* [13]. A personalization strategy emphasizes interaction between knowledge workers. The knowledge itself is kept by its creator. One personalization strategy is to record who knows what, as e.g., in yellow pages. Each person then has his/her own way to structure the knowledge. The threshold to participate is usually low, but the effort to find useful information is higher. In a codification strategy, knowledge is codified and stored in a repository. The repository may be unstructured (as in wikis) or structured according to some model. In the latter case, the structure of the repository can be used while querying. An advantage of a structured repository is that the information has the same form. A disadvantage is the extra effort it takes to cast the information in the form required. A *hybrid strategy* may be used to have the best of those different worlds [14, 15].

10.1.2 From Closed to Open Virtual AK Communities

When we speak of knowledge virtual communities we are immediately brought back to the concepts of open source software communities [16] and Internet and web-based communities [17]. Both were born as *open social environments of peers*. As such, access from non-members is allowed and aspects like task assignment and work progress are delegated to the initiative of the individual.

In the early 2000s we observed the shift of the so-called *closed communities* living inside business and governmental organizations toward more open, agile practices. This shift witnessed the creation in large business organizations of hybrid communities, such as inner-source software communities created according to the same principles of OSS development, collaborating (to some extent) with external, open communities but living within the boundaries of the organization. In a similar way, with the advent of Web 2.0, principles such as "radical trust on mass-contributed contents" or "using the web as a knowledge sharing platform" [18] enterprises applied the same principles to let their employees share the organizational know-how. For example, Yakovlev [19] gives an overview of widely known Web 2.0 mechanisms that enable the autonomous creation of virtual communities of peers. Among them we find wikis (used by enterprises to aggregate input from members of various focused groups), RSS feeds (allowing community members to remain up-to-date on selected subjects), social networking (supporting autonomous community building) and folksonomies (supporting users of a social environment in collaboratively creating and managing tags to annotate and categorize content).

In summary, organizations moved from closed to open (but regulated) communities thanks to the acceptation of modern principles and the adoption of enabling technologies. The GRIFFIN virtual community provides one example of such communities. It is meant to support a community of professionals (software architects) to effectively carry out their daily work and further contribute to (and learn from) the community with its own (architectural) knowledge. A combination of strategies for knowledge codification and personalization should provide each individual with the necessary flexibility, to fit in the own working practice and to provide sufficient incentives for successful AK management.

10.2 Requirements for collaboration in a distributed environment of software architects

Within the context of architectural knowledge management, four broad topics can be identified:

- **AK sharing** focuses on methods, tools, and techniques for exchanging AK among stakeholders directly (through personalization) or indirectly (through codification).
- **AK discovery** focuses on the methods, tools, and techniques to find, extract, and make accessible the relevant AK dispersed across the documentation that accompanies a software product.
- **AK traceability** focuses on methods, processes, and tools for codifying and interrelating AK.

- **AK compliance** focuses on ensuring that the architectural design decisions are known, understood, and complied with in the resulting system.

The combination of the four topics of AK sharing, discovery, traceability and compliance poses the following requirements for collaboration in distributed environments of software architects:

Manage architectural decisions. Architecting is a decision making process and architects have to consider lots of technical and non-technical requirements, constraints and concerns. To assist architects in the thought process of balancing these forces, the collaborative virtual community needs to offer support for managing architectural decisions and all associated knowledge. This will allow sharing of the 'reasoning behind' architectural designs, because this is what architectural decisions and their rationale represent. It also allows maintaining an explicit backlog of open issues, concerns, and decisions [20]. This requirement for the collaborative virtual community includes providing overviews of architectural decisions taken, plus the relationships between those decisions. Finally, insight in the completeness, correctness, and consistency of a set of architectural decisions helps architects in reflecting on the developed solutions, and in identifying conflicts between decisions taken.

Codify architectural knowledge. The result of the processes of architecting are reported in artifacts like documents and models. Sharing them allows AK transfer. In this way the architects and stakeholders not directly involved in decision making, can participate or acquire after-the-fact information.

Search architectural knowledge. Next to assisting practitioners in producing architectural knowledge, support during consumption of such knowledge (e.g., searching) is equally important. The need for a more balanced view on AK sharing, in which support for both producing and consuming AK is included, has been discussed before [21]. Moreover, one of the main requirements practitioners stated is support in retrieving the *right* architectural knowledge at the *right* time [22]. This can boost reuse of AK (reusable assets are better accessible) and stimulate learning among practitioners (knowledge can be found more easily).

Support community building. Due to the size and complexity of most software systems, it is often infeasible for one architect to be responsible for everything himself. This focus on teamwork is especially true in global software engineering environments where the architect-role is often fulfilled by multiple collaborating architects. Consequently, AK management support should support community building. This may include facilities to hold discussions or chat with colleagues, to organize and plan meetings, workshops or events, to peer-review deliverables of colleagues, to find

contact information, expertise and interests of colleagues, and to retrieve information about what colleagues are currently working on.

Provide intelligent support. We argue that architects would welcome intelligent support (advice, guidelines) just after or during activities producing and assessing AK (e.g., writing an architectural description). Intelligent support is more useful if combined with a certain level of pro-activity. For instance, intelligence and pro-activity can be provided using avatars that think along with practitioners and suggest ideas, challenge decisions, play the devil's advocate, etc.,

Enrich architectural knowledge. Ideally architectural knowledge should be produced and shared *below the surface* without bothering architects. Automatically distilling patterns out of unstructured data, for example, would lead to production of AK without an architect explicitly doing this. Producing and consuming architectural knowledge should thus not be considered an extra, resource-consuming activity but rather an invisible part of other organizational processes. Enrichment of architectural knowledge means support for intelligence and pro-activity, which would also benefit practitioners in their daily work, is the (semi-) automatic interpretation of content in order to enrich this content. Text mining services could for example be employed to automatically sift and winnow through existing architectural knowledge stored (e.g., in a database) looking for new patterns, define best practices, or locating trends. Based on the findings additional meta-data could be generated by such a service and eventually presented to the practitioner.

10.3 A collaborative virtual community for AK management

Within GRIFFIN, we envision a virtual and distributed community of professionals willing to create and share knowledge.

A *virtual community* is defined on Wikipedia as "*a group of people that primarily interact via communication media [...] rather than face to face, for social, professional, educational or other purposes*". We extend this definition to embrace organizations as well as individuals. Accordingly, we consider a virtual community as a group of *virtual spaces*, where each virtual space can correspond to whole organizations, teams of people or individuals. As illustrated in Fig. 10.1 organizations can share AK in a grid-like configuration of connected sites (like organization A) and/or departments or business units (like organization B) where employees carry out collaborative activities. Individuals hence work in their virtual space where they can manage their own knowledge and eventually share part of

this knowledge with (remote) counterparts in a collaborative social network of professionals.

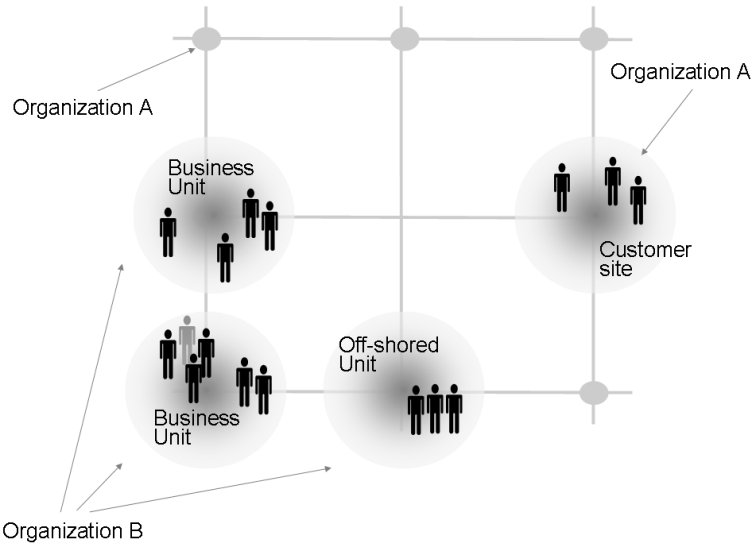

Fig. 10.1. Distributed community of organizational virtual spaces

10.3.1 Support for collaborative AK management

For each of the four AK management topics introduced before (AK sharing, AK discovery, AK traceability, and AK compliance) we researched the state-of-the-practice as well as the challenges experienced by the GRIFFIN industrial partners. For each topic, the following illustrates the related virtual spaces that we designed and developed, and the architecture process activities they support.

Virtual spaces for AK sharing

There are several broad activities within the architecting process that demand for architectural knowledge sharing (AKS). These include:

Decision making. Architecting is inherently a decision making process. Architects need to balance quality criteria, stakeholder concerns and requirements, and take a number of decision decisions in which they reuse architectural styles and patterns. Architects guide the architecting process by interacting with stakeholders, and are typically involved in various organization and business related processes. To keep track of all knowledge being created or shared in these processes, architects maintain a backlog

[20]. In this backlog an overview of decisions taken, constraints, concerns, open issues etc., are maintained to facilitate decision making, and check for conflicts or other issues.

Building up architectural knowledge. Although every software development project is unique, some architectural solutions can be applied in different circumstances. To facilitate reuse of architectural best practices (such as architectural patterns, styles and tactics) this architectural knowledge needs to be built up. This process involves transforming application-specific architectural knowledge into application-generic architectural knowledge that can be retrieved easily and applied in future projects.

Stay up-to-date. A lot of architectural knowledge is potentially relevant for architects. In order to build up expertise it is important to stay up-to-date on market trends and to be able to learn from available application-generic and application-specific architectural knowledge.

Describing software architectures. One of the important tasks of architects is writing down their solution and communicating it with their stakeholders. Often architecture design is described using a number of architectural views and viewpoints. In creating an architecture description important aspects are both the structure of the document and its completeness and internal consistency. To achieve this, annotation of AK within an architecture description is necessary.

Personalization support in a community. It is important that architects know where to find and how to contact each other when needed, so that the expertise of one architect can assist others. Services such as a chat service or yellow-pages service ("who knows what") can be used for this purpose.

To carry out these AK sharing activities, several conceptual scenarios have been designed for the AKS virtual spaces, some of which we show below:

Discuss and negotiate (Scenario $S_{AKS,1}$)
Situation: Architect(s) need to decide for an architectural design. This involves meeting all needs and concerns of the relevant stakeholders.
Problem: Each stakeholder has its own concerns and needs that often conflict with the overall goals of the system to be developed. Architects need to balance all these concerns in a satisfactory way.
Solution: With a decision making component architects are better supported in negotiating or discussing with colleagues or other stakeholders in the architecting process about which decisions to take and why. This component acts as an automated way of managing the backlog. It facilitates architects in dealing with multiple concerns at the same time by visualizing the decision space, indicating which decisions conflict with each other, etc., this also helps in personally analyzing tradeoffs and conflicts between decisions and alternative solutions. The decision making component mani-

pulates (i.e., create, read, update, delete) AK stored in a decision space database that keeps a data set for each project.

Scenario description (see Fig. 10.2):

a) The architects use the decision making component as visual guide during their discussions and negotiations about the architecture design.

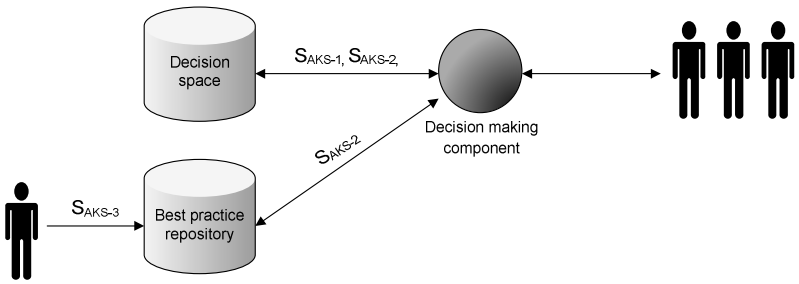

Fig. 10.2. Decision making component in relation to codified AK

Subscribe to architectural knowledge (Scenario $S_{AKS,4}$)

Situation: Architects would like to stay up-to-date.

Problem: How to inform architects of potentially available architectural knowledge without flooding them with information?

Solution: An architect can use a subscription and notification service to subscribe to specific AK topics. Based on this information the architect's user profile is created or updated. The user profiles database connects to databases where the architectural knowledge itself is being stored (i.e., the decision space database and best practice repository) to determine what types of architectural knowledge an architect can subscribe himself to.

Scenario description (see Fig. 10.3):

a) The user profiles database keeps a list of subscription topics built from the contents of the decision space database and best practice repository. These AK sources define a number of topics dependent on the AK stored.

b) Using the subscription & notification service, the architect creates his user profile by adding contact information, expertise areas and by indicating in which architectural knowledge categories he is interested.

c) All codified architectural knowledge that fits these categories is marked as potentially interesting to this subscribed user and presented to him when the time is right (cf. Scenario $S_{AKS,5}$).

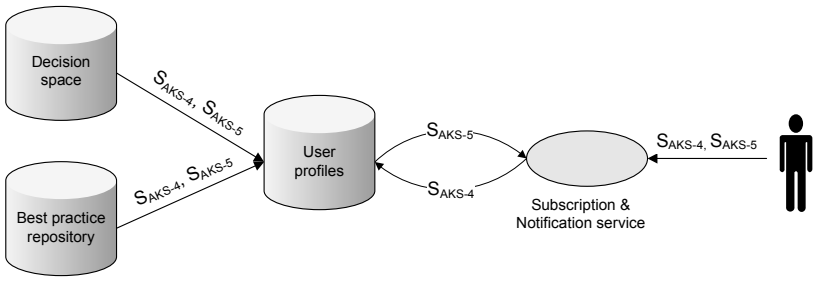

Fig. 10.3. Subscription and notification of AK

Notify architects about architectural knowledge (Scenario $S_{AKS,5}$)
Situation: Architects would like to stay up-to-date.
Problem: How to inform architects of potentially available AK without flooding them with information?
Solution: An architect is notified by a subscription and notification service about potentially interesting AK depending on his user profile (for example using RSS feeds or email) as soon as new AK is stored in one of the databases. This notification mechanism enables the Just-in-Time AK requirement discussed in [22].
Scenario description (see Fig. 10.3):
a) The subscription and notification service periodically scans for updated AK codified in one of the databases, and tries to match this with the user profiles stored.
b) The AK (or a link to the source) is pushed to all users whose profiles indicate a match.

Virtual spaces for AK discovery

Although AK discovery has broader applications, it has originally been developed and piloted to support software quality audits. Discovery of AK from software product documentation is a typical activity that an auditor must perform to collect the information necessary for expressing an opinion on a product's quality. A quality assessment entails a comparison of the SOLL-state of the software product with its IST-state. For this comparison, a thorough understanding of the actual state of the software product is obviously needed. A problem an auditor may encounter is one of *information overload:* by the time the quality of a product is being assessed, usually many documents have been written throughout which architectural knowledge is scattered. The documentation typically contains information on many different topics, including high-level system architecture, functional design, logical design, and infrastructure architecture. These topics are not confined to a single document, but have relations with other topics in other documents as well.

The result of the AKD process is the so-called *augmented documentation*, i.e., a semi-structured combination of the (unstructured) product documentation and a (structured) quality ontology that defines generic quality criteria and relations between them. The documentation is augmented with *Latent Semantic Analysis*-inferred meaning (cf. [23]) and related to applicable quality criteria selected from the quality ontology. Parts of the documentation that have a meaning closely related to the meaning of a selected quality criterion have been identified. The selected quality criteria form an index to the product documentation.

Augmented documentation eases the 'findability' of architectural knowledge and the comparison of IST-state product documentation with the SOLL-state evaluation frame. By using the LSA text analysis technique, the semantic structure underlying the product documentation can be found. This allows for suggestions regarding where to start reading when one is interested in a particular topic. It also allows for suggestions regarding how to continue reading such that the semantic difference between two consecutive documents is as small as possible, essentially providing a reading guide or a route through the documentation. Such a reading guide may for instance suggest a smooth trajectory from a high-level architectural overview to increasingly finer-grained specifications.

Some of the most important topics from a quality audit point of view are topics related to quality attributes and/or quality criteria. Therefore, in the discovery space the documentation is related to the quality criteria from the quality ontology. Parts of the documentation that have a meaning closely related to the meaning of a selected quality criterion are identified through LSA. The selected quality criteria form an index to the product documentation. Since the quality ontology defines relations between quality criteria, relations between product documentation parts can be inferred.

To carry out the AKD activities here described, the following scenarios have been supported by the AKD virtual spaces (shown in Fig. 10.4).

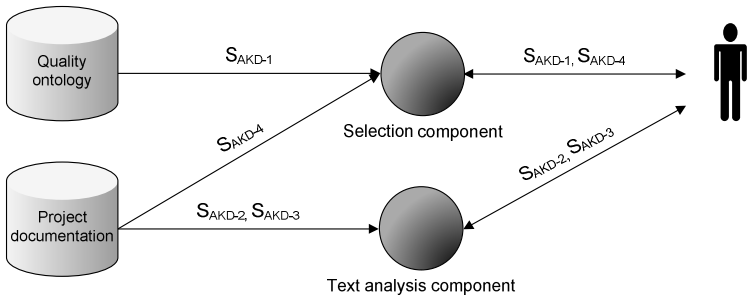

Fig. 10.4. AK discovery in quality audits

Selection of quality criteria (Scenario $S_{AKD,1}$)
Situation: Start of the audit, where quality attributes and their priorities (according to the customer) are known.
Problem: Which quality criteria to use to assess the product's compliance with the customer's requested level of quality? Since quality criteria are applicable in different product audits, auditors may read through previous audit reports to find out which quality criteria can be used. Obviously, such ad-hoc reuse is far from ideal, being time consuming and not transparent.
Solution: Codification in 'quality ontology' of quality criteria and their relations according to generic AK structures (e.g., Kruchten's ontology [24]) makes them available for more systematic reuse. Intelligent visualization supports the auditor in deciding which criteria to use.
Scenario description (see Fig. 10.4):
a) The auditor uses the Selection component to provide a list of prioritized quality attributes (e.g., 1=performance, 2=security, 3=usability).
b) The auditor is presented with a list of measures that are known to favor those quality attributes (e.g., 'use secure connections' for security) or to hinder them (e.g., 'don't use passwords'). From those measures, auditors may derive quality criteria: measures that they expect to be in the product.
c) The auditor indicates which measures should and should not be in the product, i.e., selects the quality criteria to be used in the audit. Since certain measures may be related (e.g., be in conflict or depend on each other) certain combinations are not allowed and some others are mandatory. The quality ontology identifies inconsistencies in the selected criteria and provides suggestions to solve them.
d) Further decision support is provided through mining from previous audits latent relations that are not (yet) codified. This leads to suggestions such as 'auditors who selected the criterion you just selected, also selected criterion X'.

Accessing the body of knowledge (i.e., where to start reading, scenario $S_{AKD,2}$)
Situation: quality criteria have been selected; auditors need to read the product documentation to gain a certain level of knowledge about the product they are auditing. They want to gain a high-level understanding of the most important parts of the product, i.e., 'the architecture'.
Problem: the auditor does not know where to start reading, due to information overload (too many documents) and AK scattered across multiple documents.
Solution: Text analysis (LSA) discovers the semantic structure of the set of product documents and relates the meaning of high-level words (e.g., 'architecture') to relevant parts in the product documentation, even if those words are not actually used in that text (cf. [6]).

Scenario description (see Fig. 10.4):
a) The auditors determine the type of information they need and provide a term that denotes this interest (e.g., 'architecture').
b) The auditors are provided with a list of documents (or parts of documents) ranked according to how close the meaning of the text is to the meaning of the term the auditors provided (cf. [6]).

Guidance through the body of knowledge (Scenario $S_{AKD,3}$)
Situation: The auditors have read part of the documentation and want to continue gaining further insight for the audit.
Problem: The auditors want to have a smooth progression through the documentation, however, without any big jumps from e.g., high-level overview to low-level detail and back again.
Solution: Text analysis provides a distance measure between different text parts that is employed to guide the auditor through the documentation.
Scenario description (see Fig. 10.4):
a) The auditors determine their subsequent information need and provide a corresponding term (e.g., the name of a module for further investigation).
b) The auditors are provided with a list of documents ranked according to: how close the meaning of the text is to the meaning of the provided term; and how close the meaning of the text is to the meaning of the previously read text (cf. [6]).

Quality assessment (Scenario $S_{AKD,4}$)
Situation: The auditors have gained an overall understanding of the software product and now need to determine the product's compliance with the selected quality criteria.
Problem: Again, information overload: Too many documents and not all information regarding a particular product quality can be expected to be located at a single place.
Solution: By relating the meaning of the quality criterion (as defined in the quality ontology by its description and relation to other criteria) to the meaning of the software product documentation, parts of the documentation that talk about the criterion can be identified.
Scenario description (see Fig. 10.4):
a) The auditors select a criterion that they want to investigate.
b) The auditors are provided with a list of documents ranked according to how close the meaning of the text is to the meaning of the quality criterion.

Virtual spaces for AK traceability

In AK traceability, three concepts play an important role:

- ***Concepts:*** The classes of distinguished AK.
- ***Relationships:*** The relationships among these classes.

- ***Knowledge Entities (KE):*** Instances of a particular concept that can have relationships to one or more other instances.

The activities in a virtual space for AK traceability use these concepts. They include the following activities:

Identify AK and traceability needs. Codifying AK and providing traceability at the same time is a costly operation. Hence, it is important to minimize the required effort to do so. This is achieved in two ways: by focusing on the real AK needs and by reducing the effort of capturing of creating traceability.

Modeling the required AK and traceability information in a domain model. Based on the identified needs, the virtual space should assist an architect in defining a domain model for modeling the relevant AK concepts and relationships. This can take the form of suggesting (part-of) existing models based on the earlier identified needs.

Capture the knowledge according to this domain model. The virtual space should assist stakeholders with capturing the relevant AK in KEs. This is achieved by either automating the process (such as investigated in the discovery virtual space) or by offering intelligent integrated tooling in environments in which this knowledge is created or described.

Integrate captured knowledge with other sources. For the virtual space to offer optimal traceability, the captured knowledge should be integrated (i.e., related) to knowledge of other relevant sources. This activity is often intertwined with the capturing activity. There are several ways in which a virtual space could achieve this integration. First, a virtual space could automate this integration, e.g. by using text analysis techniques. Second, it could offer step-by-step suggestions on how this integration could take place, thereby guiding the integration process. Third, it could offer search functionality and associated suggestions to facilitate a manual integration process. Often, a combination of these three different possibilities is used for different concepts.

Consume the AK and its traceability. Once the needed AK has become traceable, this knowledge can be used for various purposes, including the production of additional AK and the identified AK and traceability needs. Some of example scenarios of this usage will be presented.

Evolve the knowledge. Typically, architecture is designed in multiple iterations. Hence, there is a need to not only evolve the architecture design, but also its associated knowledge and relations. A virtual space should support incremental updating of the KE and relationships, both in a reactive and proactive manner. For the former, a stakeholder wants to change some AK, and see the consequences of this change. For the latter, a virtual space should be able to detect certain changes and evolve related AK accordingly. For example, the removal of a requirement potentially invali-

dates the architectural decisions based on this requirement. With traceability, a virtual space could automatically determine such impacts.

Find specific AK to relate to (Scenario $S_{TA,1}$)
Situation: The software architect wants to find specific AK to relate to, so as to create traceability among the KE.
Problem: The number of KEs is typically very large and the specific KE might not be codified yet.
Solution: Based on the domain model, the virtual space makes a first selection of KE that could be related. Hence, it acts as a classification filter. In addition, the virtual space uses the traceability information of the starting KE as a way to guess what the context of the start point is and use this information to assist in the search process.
Scenario description:
a) The software architect selects a KE as a starting point.
b) Optionally, the architect selects a possible relationship (automatically inferred from the domain model) for the selected KE.
c) Optionally, the architect can insert some keywords describing the KE to search for.
d) The virtual space tries to find plausible candidate KE to relate to and orders the search results.
e) The architect uses the traceability information to navigate through the search results.
f) The architect selects one of the found KE and codifies the relationship or decides to manually create the missing KE.

Make an architectural decision (Scenario $S_{TA,2}$)
Situation: The software architect wants to make an architectural decision.
Problem: The software architect needs to rationalize this decision to convince stakeholders of its relevance and correctness.
Solution: The architect defines the traceability of the architectural decision to other AK. This makes the rationale of the decision traceable and helps in making the decision process more transparent.
Scenario description:
a) The software architect, helped by the virtual space, scopes the problem space, thereby identifying the reason why an architectural decision has to be made.
b) The software architect defines the alternative(s) considered.
c) The architect captures the evaluation of the alternatives. The rationale for a particular choice is codified by providing traceability to specific AK from the problem space.
d) The impact of the chosen alternative is considered for both the problem and domain space. New AK is created and related accordingly.

Design maturity assessment (Scenario $S_{TA,4}$)

Situation: The software architect wants to know how mature the software architecture design is. This includes the correctness, completeness, and consistency of the design and its description.
Problem: Judging the maturity of a design is not trivial, as it requires harmonizing subjective judgments of multiple experts on both individual and collections of KE. Again, information overload: Too many documents and not all information regarding a particular product quality can be expected to be located at a single place.
Solution: The traceability provided by the codified AK allows for an automated assessment of the completeness of the AK. Since the defined domain model allows for assumptions about AK that should exist and their relationships. The explicit AK provides stakeholders the opportunity to assert and administrate the correctness of each individual KE. Consistency is improved, since navigating through and finding related AK becomes more easy thanks to increased traceability.
Scenario description:
a) The software architect selects an architecture description the maturity should be assessed of.
b) The virtual space identifies which parts of the AK are incomplete.
c) The architect completes these AK omissions.
d) The architect shares the architecture description and associated AK with relevant stakeholders.
e) Each of these stakeholders asses the correctness and consistency of the AK and identify in the virtual space which parts are troublesome.
f) The architect collects these remarks through the virtual space and resolves them in a new version of the architecture description.

Virtual spaces for AK compliance

The architecture of a software system guides the software development activities by providing the necessary direction for it. Architectural rules are the principles and statements on the software architecture that must hold at all times, and thus must be complied with [2]. Architectural compliance in global software development (GSD) environments poses additional challenges for sharing AK and complying with architectural rules.

The aim of compliance verification is that the resulting system is in line with the principles as expressed in architectural rules. A collaborative virtual space should allow for continuous compliance verification by promoting architectural knowledge to relevant stakeholders and development sites to reduce the gap between reality and the principles identified during compliance verification. Hence, the virtual space for AK compliance should not only support compliance verification in hindsight, but partly overlap with the virtual space for AK sharing.

To ensure compliance in GSD environments, the virtual space for AK compliance supports the following activities:

Identify architectural rules requires the virtual space to characterize a (possible sub-) set of architectural design decisions that are mandatory. The virtual space presents the architectural design decisions in a format which allows practitioners to perform compliance verification, by allowing to indicate entry criteria for e.g. the applicability of architectural rules for only part of the system, and criteria that allow practitioners to determine when architectural rules are satisfied, and when they are not.

The *"inject architectural rules in company"* practice is necessary to let the architectural rules sink in within the organization. The architectural rules need to be made known to the practitioners across the different development sites and their understanding should be verified explicitly.

Verify compliance supports matching designated parts of the implemented system with applicable architectural rules. The virtual space for AK compliance further supports a compliance officer in this process by running compliance checks automatically, when applicable. The compliance verification results in a list of non-compliance items that indicate what architectural rules are not complied with and where in the system this non-compliance occurred.

Address situations of non-compliance The results of the verification are interpreted by the compliance officer and presented to the software architect(s). The virtual space for AK compliance indicates the severity of the non-compliance which helps the software architect to take adequate follow-up measures. These follow-up measures can pertain to instructing or re-implementing architectural rules within the software architecture, or for adjusting the set of architectural design decisions which, in turn, will affect the set of architectural rules that hold.

The virtual space for AK compliance supports the following scenarios:

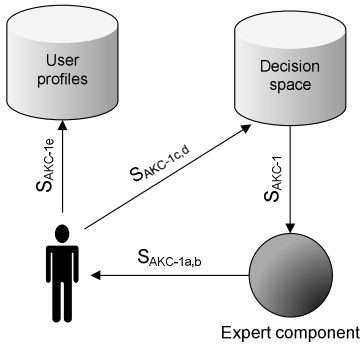

Fig. 10.5. Identify architectural rules

Identify architectural rules (Scenario $S_{AKC,1}$)
Situation: Architect(s) need to decide what architectural knowledge should be complied with in the software.
Problem: How does an architect indicate what architectural knowledge is mandatory? How can an architect be supported in providing the correct information that allows for both correct implementation and compliance verification?
Solution: Designate a subset of architectural design decisions as architectural rules.
Scenario description (see Fig. 10.5):
a) The architect is provided with the set of architectural design decisions from the decision space.
b) The architect selects a set of architectural design decisions that should be complied with.
c) The architect augments the architectural design decisions with knowledge necessary to increase their 'verifiability'. This includes e.g.,

- Identification of the scope (both related to the system and the project) and the impact of non-compliance.
- Identification of the way compliance verification can take place (using e.g., automated tools or manual inspections).

d) The architect identifies a compliance verification method from a list of verification options provided to him.
e) The architect identifies the stakeholders (per development site) that need to be informed of the AK to comply with.

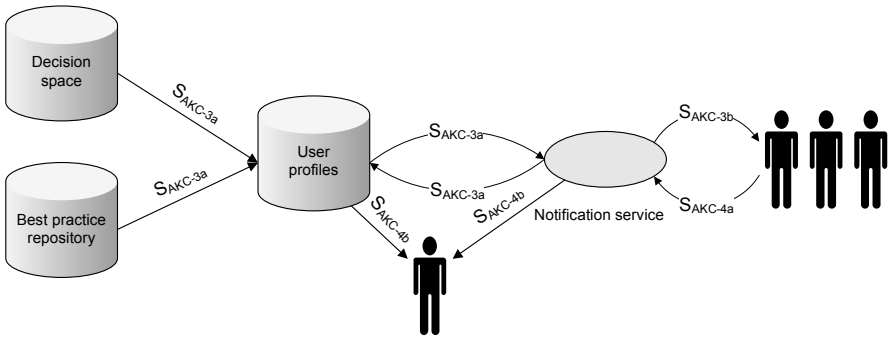

Fig. 10.6. Push architectural rules to stakeholders and verify their understanding

Push architectural knowledge to relevant stakeholders (Scenario $S_{AKC,3}$)
Situation: Relevant stakeholders of architectural knowledge need to know what architectural rules are mandatory and need to be complied with.

Problem: How to ensure that all relevant stakeholders are informed of the architectural design decisions?
Solution: Use a notification system (see Scenario $S_{AKS,5}$) and ensure that stakeholders have consumed the architectural knowledge. The solution does not make use of a subscription service (Scenario $S_{AKS,4}$) but uses a predefined set of stakeholders that must be informed.
Scenario description (see Fig. 10.6):
a) The notification service matches architectural knowledge designated as architectural rules with the user profiles.
b) Based on the user profiles that need to be informed, the notification service provides the architectural rules to the corresponding users.

Verify understanding with AK (Scenario $S_{AKC,4}$)
Situation: Relevant stakeholders need to understand the architectural knowledge.
Problem: How to ensure that all relevant stakeholders understand the architectural rules that must be implemented or complied with?
Solution: It is important to obtain feedback from the relevant stakeholders on their understanding of, or concerns regarding this architectural knowledge. When development sites are distributed, effective implementation of AK can only occur by collecting feedback from these development sites [2, 4, 25].
Scenario description (see Fig. 10.6):
a) Practitioners who have received the architectural rules can indicate whether they are informed of the architectural knowledge.
b) Feedback on the AK is solicited and transferred to the architect.

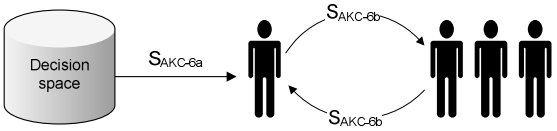

Fig. 10.7. Provide follow-up to compliance results

Address situations of non-compliance (Scenario $S_{AKC,6}$)
Situation: A system does not comply with the current architectural rules.
Problem: What are possible measures that the architect can take?
Solution: The architect can either identify if the current architectural rules must be modified to accommodate the current situation, or the practitioners of the responsible development sites need to change the system comply with the architectural rules.
Scenario description (see Fig. 10.7):
a) The architect decides that certain architectural rules in its original form deemed inapplicable and updates the architectural rules accordingly.

b) The architect reinforces the existing architectural rules. The architect may use scenarios $S_{AKC,3}$ and $S_{AKC,4}$ as a minimum.

10.3.2 Towards a virtual AK sharing community

The previous sections presented the conceptual scenarios supporting AK sharing, discovery, traceability, and compliance in a distributed virtual space. Let's imagine an AK sharing community of networked member organizations, each supporting one or more of such scenarios. In addition to their individual contribution, each scenario provides generic features that can further propel collaboration, which is called 'social cognition' in [25] i.e., *"the ability of a group of people to remember, think and reason"*.

For example (see Fig. 10.8) an auditing organization can locally carry out the quality audit of a product developed by a certain customer organization. The auditing organization, on its own, can locally annotate AK, which might be relevant for that audit. If the auditing organization and the customer organization connect their local virtual spaces and if relevant auditors can subscribe to and be notified of relevant new AK annotations, auditors are able to speed up the learning process about what knowledge is necessary to achieve an opinion about the product's quality. Further, experience and know-how can be improved, as well as the level of trust between the two partner organizations.

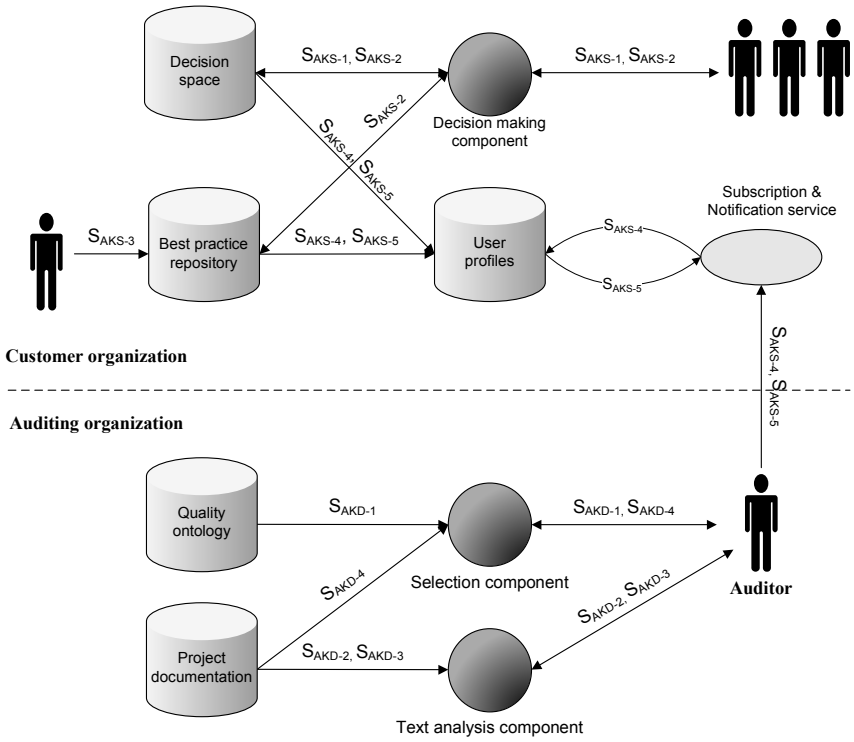

Fig. 10.8. Towards a community: connecting virtual spaces

In order to provide more advanced AK management support we envision more of these scenarios that involve connecting virtual spaces of different organizations or departments of organizations. This will further enhance collaboration among different parties and will help in increasing the virtual community of architects.

10.4 Future trends and research challenges

Building a virtual community into an organization is a long-term investment and introduces substantial change. We need to bring convincing arguments, backed by hard data, that such an investment is worthwhile. We also need ways to realize such migration. Also to ensure that new scenario combinations (such as the example discussed in Sect. 3.2) improve the state of the practice, a research challenge is to obtain a better understanding of what practitioners in the architecting process need.

A second research challenge is related to the different terminology used by different organizations. Different organizations speak their own “language” of AK. If AK is to be shared between organizations, then the vir-

tual collaborative community needs to support appropriate translations from the AK meta-model of one to those of the other virtual spaces. This is a purely technical problem and can be resolved with different technologies, e.g., from the ontologies and the semantic web community. A cost-benefit analysis must be conducted, to make the right trade-off between the cost of the translation (especially with evolving AK meta-models) and the perceived benefit (quality of the translation).

Another – more technical – challenge is the visualization of AK in the different virtual spaces. There is no one-size-fits-all visualization solution. Therefore we need customizable solutions that can be tailored to the AK meta-model and even the intended usage.

Crowd sourcing is another trend that may have a large impact on virtual AK communities. The users of these communities may scale up to thousands, and may be given the power to define, on their own, requirements and use cases for AK; they may even design their own virtual spaces. This challenge needs to be addressed both technically (provide the right crowd sourcing technologies) and non-technically (showing people the benefits and leveraging their self-motivation).

Lastly, sharing AK through the virtual organizations raises many complicated legal issues, with respect to intellectual property rights. Of course sharing AK can happen both in open and in inner (closed consortia) communities. These aspects need to be thoroughly inspected before large corporations are convinced to contribute and share AK. Also, further research is needed about creating incentives for architectural knowledge sharing, since the success of the virtual community is largely determined by the amount of time and energy the users are willing to spend on it.

10.5 Conclusions

This chapter presented the conceptual view of the GRIFFIN collaborative community for AK management. This community consists of virtual spaces supporting four key AK management topics: AK sharing, discovery, compliance, and traceability.

We discussed how each of the scenarios has been designed in the GRIFFIN project. We further illustrated one example about how such scenarios can be potentially combined to implement more complex scenarios. In this way, scenarios can provide general solutions to common AK management problems and propel collaboration among individuals and across organizations.

We would like to especially encourage the industrial community to actively participate in addressing the challenges and forming the future virtual AK communities. We have come to the understanding that in the context of global software development, the industry of software-intensive

systems faces these challenges intensively and with an increasing pace. There are still many problems that need to be resolved and there will be substantial research conducted before AK virtual communities become a reality. We hope that the industry will be keen in enthusiastically participating to this research and shape the way AK communities will collaborate in the future.

Acknowledgements

This research has been partially sponsored by the Dutch Joint Academic and Commercial Quality Research & Development (Jacquard) program on Software Engineering Research via contract 638.001.406 GRIFFIN: a Grid for Information about architectural knowledge.

References

[1] Bass L, Clements P, Kazman R (2003) Software Architecture in Practice. (Ed.), SEI Series in Software Engineering. Addison-Wesley
[2] Clerc V, Lago P, Van Vliet H (2007) Assessing a Multi-Site Development Organization for Architectural Compliance. In 6th Working IEEE/IFIP Conference on Software Architecture (WICSA)
[3] Clerc V, Lago P, Van Vliet H (2007) The Architect's Mindset. In 3rd International Conference on the Quality of Software Architectures (QoSA)
[4] Clerc V, Lago P, Van Vliet H (2007) Global Software Development: Are Architectural Rules the Answer? In International Conference on Global Software Engineering (ICGSE)
[5] De Boer RC et al. (2007) Architectural Knowledge: Getting to the Core. In 3rd International Conference on the Quality of Software Architectures (QoSA)
[6] De Boer RC, Van Vliet H (2008) Architectural Knowledge Discovery with Latent Semantic Analysis: Constructing a Reading Guide for Software Product Audits. Journal of Systems and Software, (81) 9:1456-1469
[7] Farenhorst R, Lago P, Van Vliet H (2007) EAGLE: Effective Tool Support for Sharing Architectural Knowledge. International Journal of Co-operative Information Systems (IJCIS) Vol. 16, (3) 4:413-437
[8] Jansen A et al. (2007) Tool Support for Architectural Decisions. In 6th Working IEEE/IFIP Conference on Software Architecture (WICSA)
[9] Jansen A et al. (2008) Sharing the Architectural Knowledge of Quantitative Analysis. In 4th International Conference on the Quality of Software Architectures (QoSA)
[10] Bosch J (2004) Software Architecture: The Next Step. In First European Workshop on Software Architecture (EWSA)
[11] Zhuge H (2004) The Knowledge Grid. World Scientific Publishing Co
[12] Nonaka I, Takeuchi H (1995) The Knowledge-Creating Company. Oxford University Press

[13]Hansen MT, Nohria N, Tierney T (1999) What's Your Strategy for Managing Knowledge? Harvard Business Review, (77) 2:106-116
[14]Ali Babar M et al. (2007) Architectural Knowledge Management Strategies: Approaches in Research and Industry. In 2nd Workshop on Sharing and Reusing architectural Knowledge - Architecture, rationale, and Design Intent (SHARK/ADI)
[15]Desouza KC, Awazu Y, Baloh P (2006) Managing Knowledge in Global Software Development Efforts: Issues and Practices. IEEE Software, (23) 5:30-37
[16]Capiluppi A, Lago P, Morisio M (2003) Characteristics of open source projects. In European Conference on Software Maintenance and Reengineering
[17]Preece J (2000) Online Communities: Designing Usability, Supporting Sociability. Wiley
[18]O'Reilly T (2005) What is Web 2.0 - Design Patterns and Business Models for the Next Generation of Software
[19]Yakovlev IV (2007) Web 2.0: Is It Evolutionary or Revolutionary? IT Professional, IEEE Computer Society, 9:43-45
[20]Hofmeister C et al. (2007) A General Model of Software Architecture Design Derived from Five Industrial Approaches. The Journal of Systems and Software, (80) 1:106-126
[21]Lago P, Avgeriou P (2006) 1st Workshop on Sharing and Reusing Architectural Knowledge, Final Workshop Report. ACM SIGSOFT Software Engineering Notes, (31) 5:32-36
[22]Farenhorst R et al. (2008) A Just-In-Time Architectural Knowledge Sharing Portal. In 7th Working IEEE/IFIP Conference on Software Architecture (WICSA)
[23]Landauer TK, Foltz PW, Laham D (1998) An introduction to latent semantic analysis. Discourse Processes, 25:259-284
[24]Kruchten P (2004) An Ontology of Architectural Design Decisions in Software-Intensive Systems. In 2nd Groningen Workshop on Software Variability Management
[25]Chi E H (2008) The Social Web: Research and Opportunities. IEEE Computer (September) pp 88-91