Tool Support for Automating Architectural Knowledge Extraction

Aman-ul-haq, Muhammad Ali Babar
Lero, University of Limerick Ireland
Aman.ul.haq@lero.ie, malibaba@lero.ie

Abstract

Development of large projects is a knowledge intensive task. Applying knowledge management techniques to project activities can enhance productivity and reduce risks of failures. However, it has been observed that knowledge management activities suffer from problems such as unavailability of structured information and lack of incentives to put extra efforts for these activities. In this paper, we present a tool that captures architectural knowledge from documents and emails and stores it in more structured manner in knowledge repositories with minimum user intervention, thus minimizing the required amount of effort.

1. Introduction

The development of large and complex system is likely to suffer from communication problems among the stakeholders involved in making key technical as well as non-technical decisions. A project may fail as a result of poor communication and unavailability of information surrounding key decisions. For example, in the case of NASA's Mars Climate Orbiter [1], it was reported that a good communication between navigation and development teams could have avoided the disaster despite the root cause of the failure was a mistake in the navigation software. It was revealed that the navigation team failed to communicate their concerns to the development team even though they were aware of errors in the trajectory estimation of the spacecraft [1]. Recent efforts to address these kinds of problems in the area of software architecture are focused on managing Architectural Knowledge (AK).

The knowledge surrounding software architecture design decisions is called AK [16]. Architectural decisions and design options form the most important parts of AK. On one hand, they represent the decisions taken; on the other hand, they capture the rationale for these decisions and help us to reason about different quality attributes at the architecture level. Realizing the importance of AK, many organizations have started paying more attention towards codification of tacit knowledge underpinning their architectural processes and artefacts [2]. Architectural information is usually documented in lengthy documents which pose problems such as: 1) locating relevant information inside a long document is time consuming and difficult task; and 2) traceability among different architectural elements is lost. To overcome these problems, research community has developed several Architectural Knowledge Management (AKM) tools such as PAKME [7], ADDSS [18], Archium [4] and AREL [5]. Nevertheless the applicability of these tools to real industrial settings appears to be quite challenging because of the following reasons:

1- While these tools provide systematic ways of capturing and managing AK, industry is likely to prefer their conventional approaches to sharing and managing AK (i.e. via documents and emails).

2- Manually transferring AK from documents to knowledge repositories is laborious and painstaking.

This tension between lack of available time and pressure to keep knowledge repository current usually results in inconsistent and incomplete AK captured in an organizational knowledge repository. Hence, our research goal is to find an effective and efficient way of capturing AK, which not only relieves an architect from unnecessary effort for capturing and sharing AK but can also be easily incorporated into existing processes and productivity tools commonly used. Our assertion is that if there is a tool that automatically extracts the required AK from documents and emails and semi-automatically enters that AK in an organizational knowledge repository, it can improve the likelihood of repository-driven AKM tools being widely used. In this context, we have developed a tool called *Automatic Architecture Knowledge Extraction Tool* (AAKET), which is expected to perform most of the time-consuming tasks semi-automatically with minimum human intervention. The tool is also expected to help organizations to maintain their knowledge repositories current and consistent with relatively less effort.

978-1-4244-3726-9/09/$25.00

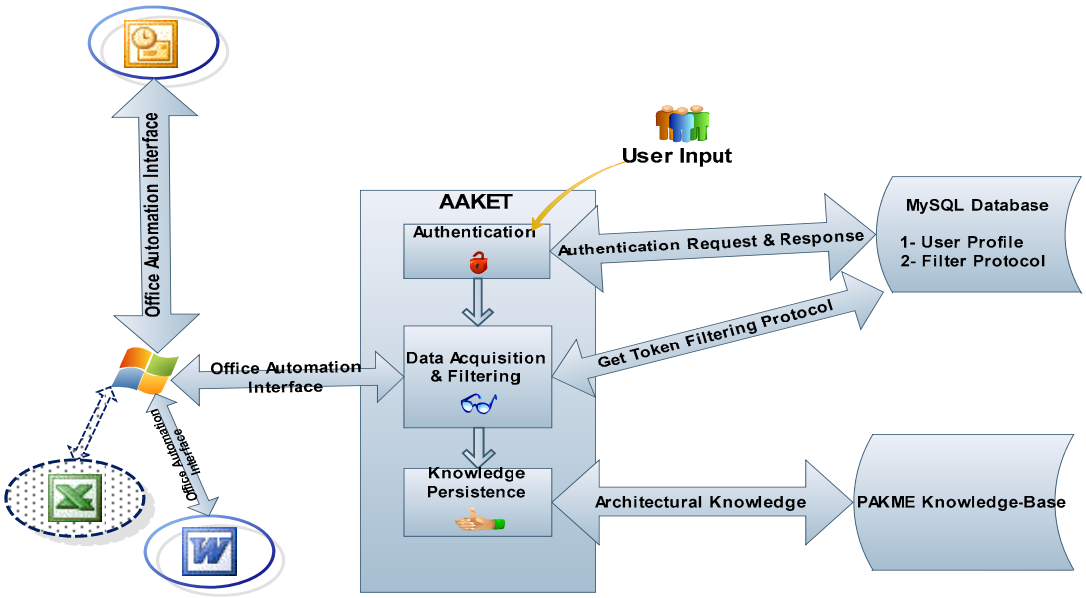

Figure 1. High level architectural view of AAKET

The remainder of this paper is organized as follows: Section 2 outlines and explains the architecture of the tool, Section 3 focuses on implementation aspects and different technological choices, Section 4 presents preliminary evaluation of our tool, Section 5 discusses the related work and Section 6 outlines the possible future enhancements.

2. Architectural Details

The high level architectural view of AAKET has been shown in Figure 1. AAKET's architecture has been designed as layers of components responsible for performing various functions provided by the tool. Each component plays its role independently. The components communicate with each other through well-defined interfaces. That is why AAKET's architecture is capable of accommodating future enhancements in any part of the tool without affecting other parts. Following are the main components of AAKET:

Authentication: This component has been introduced to assure the security of the information stored in a repository. Since several repositories can be populated using AAKET's services, it is important to ensure that users can only add new knowledge to a repository for which they have been authenticated. That is why AAKET handles the authentication information by storing all the authentication related information in a remote server. Every organization may have separate policy of authentication (group-based, individual or global) which can be controlled from a central server.

Data acquisition and filtering: This module provides the key functionality of AAKET. It extracts the data stored in emails and other documents based on a set of filtering protocols and rules, and hands it over to the next layer component for persistence of this knowledge. This component has been developed using Microsoft office automation techniques. At this stage, we have developed the automation facilities for Microsoft Outlook and Microsoft Word. However, this component can be easily extended to provide the same features for other productivity software like PowerPoint and Excel Spreadsheet. Such an extension is already underway.

Knowledge persistence: Once the information making up AK has been extracted from e-mails and/or documents, next step is to store this knowledge in a knowledge repository. AAKET can store the extracted knowledge to any repository using an appropriate API. In our case, we have been using PKAME (Process-based architecture knowledge management environment) [7] that we have developed for managing architectural knowledge for supporting the architecture process involving geographically distributed stakeholders. We have chosen PAKME to demonstrate the use of AAKET because our industry collaborator for trialing PAKME raised the concerns about the effort required to populate the repository with the knowledge. They proposed to develop a feature that can provide partial automation of the task of capturing AK [3].

Moreover, this issue was also mentioned by Kruchten during a discussion about the use of repository driven AK management tools like PAKME [15].

3. Implementation Details

AAKET has been developed taking the performance factors into consideration. Our choice of the programming language for implementing the core components of AAKET was based on the performance requirements. We anticipate that there may be hundreds of emails that AAKET may have to process in a batch mode. Following were the programming languages that we considered for Implementation AAKET:

1. Visual Basic 6.0
2. C# .NET
3. Visual C++ 6.0

All three languages are based on Microsoft technologies as our research and development team had extensive expertise and experience in these programming languages. Both Visual Basic 6.0 and C# .NET suffer from performance issues because of their native code. Native code is interpreted by language runtime which causes some performance bottlenecks especially in CPU intensive tasks such as information extraction. Because of the abovementioned reasons, we decided to use Visual C++ 6.0 for AAKET's implementation. Secondly, we are not using any third party library that could cause any performance penalty. Our direct utilization of the MSOffice automation interface further helped us to achieve our performance goal.

Table 1. Time Complexity of popular information searching algorithms

Algorithm	Expected Running Time
Brue Force	O (nm)
Rabin-Karp	O(n+m)
Boyer-Moore	O (n/m)
Kanuth-Morris-Pratt	O(n+m)

To measure the overhead of an algorithm we used a method called "complexity calculation". There are two kinds of complexities:

1. Space Complexity: How much memory an algorithm uses during execution?
2. Time Complexity: How much CPU time an algorithm takes during its execution?

In real world situation there is always a compromise between space and time complexity. Because of the availability of cheaper memory chips, we focused on time complexity in our algorithms.

During information extraction lots of parsing is required which poses a major performance challenge. Finding the best algorithm to reduce information parsing is one of the most important requirements. Brute force algorithms for information parsing require O(nm) time where 'n' is length of whole information and 'm' is the length of the chunk to be parsed. We used Boyer-Moore algorithm whose running time is O(n/m). Because this algorithm is quite efficient, it is also used in many plagiarism detection applications and text editors. Table 1 lists the time complexity (running time) of some of the most popular string matching/searching algorithms.

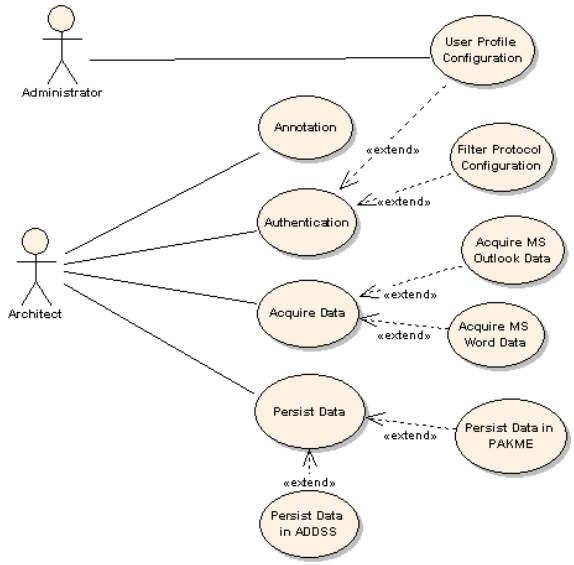

Figure 2. Use Case Model for AAKET

4. Use of the Tool

Now we explain the entire flow of using the tool for tagging AK and filtering and extracting the tagged knowledge from a text document. Figure 2 shows a Use Case Model representing the interactions between AAKET and its users. The Use Case Model shows that an architect has access to AAKET's functionalities such as annotating AK, authentication, data extraction and data persistence, but modification of user profile and annotation protocol can only be done by remote administrator.

First of all, an architect is required to annotate the information considering AK contained in MS-Outlook emails or MS-Word documents. To facilitate the architect to mark the required knowledge entities with tags, we have developed plug-ins for MS-Word and MS-Outlook shown in Figure 3 (Please note we are showing only two tags but we have been developing other tags as well such as *Rationale* etc.).

These plug-ins have been developed using Microsoft's Visual Basic for Application (VBA) technology. An Architect is only required to highlight the text and click on the respective toolbar button to mark it as either "Architecture Decision" or "Design Option".

We illustrate the working sequence of AAKET by involving two members of an architecture development team. For this illustration, let us assume that these members are James and Aman.

Step 1: James annotates the architectural information in the email he intends to send to his colleagues on some of the key architectural design decisions he has made. He tags the architectural decisions and their respective design options using the tagging button provided on his email client toolbar shown in Figure 3. The results of his tagging are shown in Figure 4, which shows that some text is highlighted in Yellow, the architectural decisions, some text is highlighted in blue, the design options. James sends this email to Aman, who is working on the same project but is located at a different place.

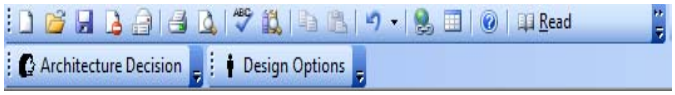

Figure 3. Plug-in for tagging architecture knowledge

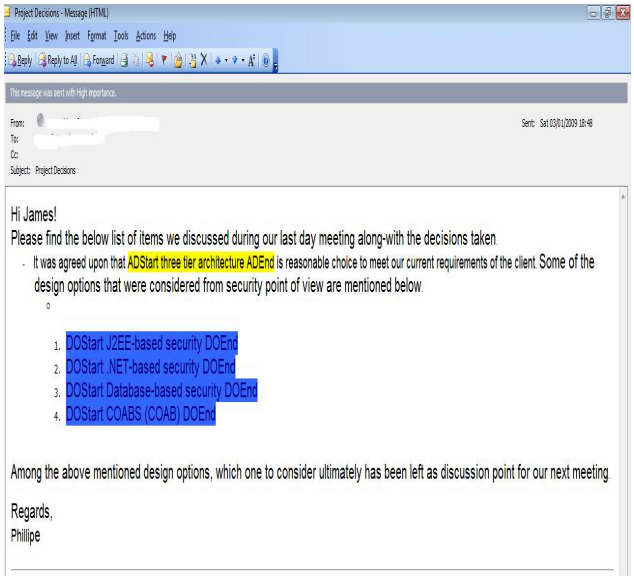

Figure 4. Annotated architectural information

Having received the email from James about the architectural design made by James, Aman decides to capture the tagged content of the email, i.e., AK, in the organizational knowledge repository. To take advantage of the knowledge extraction and storing features provided by AAKET, Aman launches AAKET tool that has been installed on his laptop for this purpose and performs rest of the activities in the following order:

Figure 5. Authentication Interface

Step 2: Aman has to get himself authenticated via a remote server to ensure the protection of the remote knowledge repository as shown in Figure 5. For this purpose, he needs to provide the URL of the knowledge repository and his username and password for that repository.

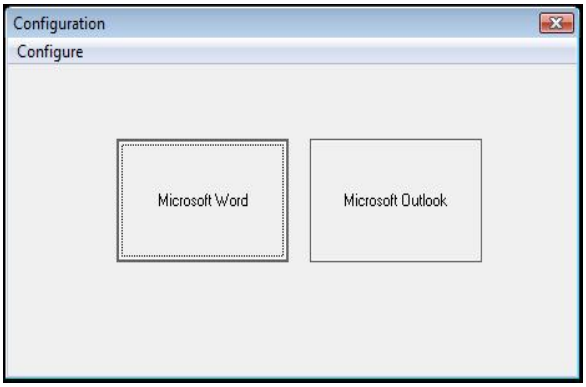

Figure 6. Application Selection Interface

Step 3: Once Aman has successfully been authenticated and connected with the remote knowledge repository, an interface selection Windows opens up with two options of scanning either e-mails (Outlook) or documents (Word) as shown in Figure. 6. Here, we assume that Aman selects the option of scanning the content of his

emails based on the current status of information he has on his machine.

Step 4: Once the selection of the interface has been made, AAKET scans the document/e-mails and presents this information to Aman as shown in Figure 7.

Step 5: Now Aman can select the pieces of information he wants to store in the knowledge repository (in our case PAKME). This step finishes the process of extracting AK knowledge from emails or Word documents and captures it in a repository.

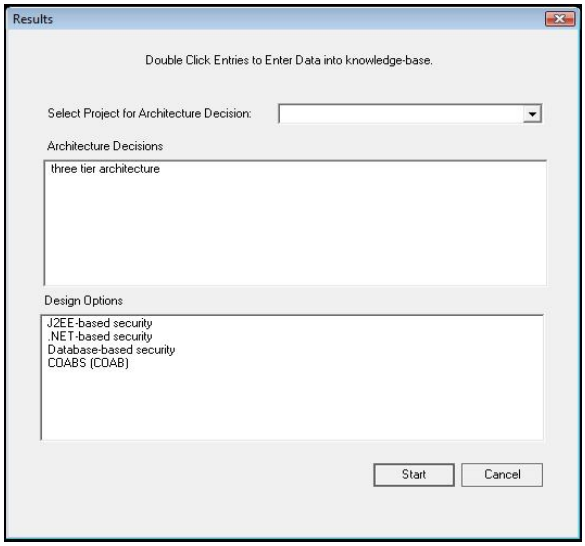

Figure 7. Knowledge Extraction Results

5. Preliminary Evaluation

We assert that tool development goes well beyond the boundaries of just an appropriate design and implementation. It is important to rigorously evaluate a tool with respect to the claims made about the expected utility and benefits of a tool. We are designing an empirical study to evaluate the benefits and limitations of AAKET as complementary mechanism for capturing and sharing AK through repository driven tools. In order to refine our evaluation goals and questions, and get an initial feedback about AAKET, we have performed an observational study. This study involved four participants who used both the interface of a knowledge repository (i.e., PAKME) and AAKET to capture a few parts of AK. At this stage, we have identified three main research questions for our evaluation settings.

1) Can AAKET keep its use independent of the knowledge about the specifics of different knowledge repositories deployed in different organizations?

2) Can AAKET be easily used without requiring significant training effort?

3) How much effort can we save with AAKET?

First question is important as a user (e.g., architect) should not be forced to know about the structure of a knowledge repository and browse its complex interfaces in order to capture relevant AK manually. It will help to save the user from learning about the repository structure every time he/she needs to use a new organizational knowledge repository. The main objective is to provide a consistent interface to different knowledge repositories that an organization may have. The second evaluation question is concerned with the amount of training required to use AAKET and ease of use in terms of user friendliness and meaningfulness of its GUI. Our concern is that if GUI is not user friendly and cluttered with too many complex details, we will be just shifting the complex flow of using different repositories to tool instead of minimizing it. Another objective is to save time and effort required to capture and share AK using repository driven tools.

To gain a preliminary evaluative feedback about AAKET with respect to the abovementioned questions, we have carried out a small observational study as aforementioned. For our study, we invited four of our colleagues researching in software architecture. Three of them were PhD students and one of them a researcher (Not any of the authors of this paper). They possessed a good knowledge of the current research and practice in AKM.

For the first research question, we decided to use the observations of the research team. For the second research question, we decided to focus on the amount of training provided and the participants' feedback. For comparing the effort required to use a repository and AAKET for capturing AK, we decided to calculate the number of keystrokes and mouse clicks required for capturing one piece of AK into repository if doing it manually versus if using AAKET. The study involved the following steps.

Step 1: Each participant was given architecture decisions and design options that they had to enter during the study in PAKME's repository.

Step 2: Each of the participants was given 10 minutes training in using PAKME's interface for capturing architectural decisions and design options (Please note that giving training using all features of PAKEM will certainly require more time).

Step 3: The participants were asked to enter the given design decisions and design options manually into PAKME's repository and answer the questions about the number of mouse clicks and keystrokes required to enter the given AK in PAKME.

Step 4: After completing the first task, each of the participants was given 5 minutes training in using AAKET.
Step 5: The participants were asked to use AAKET to extract information from a given email and store it in PAKEM. They were again required to answer the questions about the number of mouse clicks and keystrokes required to store the tagged information from the given email using AAKET.
Note - All keystrokes required to enter login page information both for PAKME and AAKET were not considered as they may vary from one login to other.

Our initial findings and the participants' feedback are very encouraging. We noticed that the participants did not need to know the structure of PAKME's knowledge repository in order to capture the given set of AK using AAKET but they need to understand the structure of the templates being used by PAKEM for capturing AK. This will be further evaluated when we evaluate AAKET with different knowledge repositories. It is obvious from the study description that the participant needed only 5 minutes training for using AAKET without any problem. Their initial comments are very positive. They have identified the areas of improvement that would be taken into account while extending AAKET. We have also found that manually entering information into PAKME requires lots of keyboard usage. Whereas there are no keystrokes involved in using AAKET once the content of AK is available and tagged. That means we expect to minimize the usage of keyboard for capturing AK using AAKET. Moreover, the usage of AAKET requires almost 50% less mouse clicks as compared with entering the same AK manually.

6. Related Work

In response to the increasing realization of the importance of providing suitable tooling support for capturing and sharing Architectural Knowledge, several researchers have developed various tools [6, 8, 11, 19] for managing architectural knowledge and others have identified requirements with the intention of providing such tools [10]. One of the earliest tools for managing architectural knowledge is Archium, which models design decisions and their relationships with resulting components [4]. The Knowledge architect is another recently developed toolset for managing architectural knowledge [12]. The toolset comprises a repository, a server, and a number of clients. EAGLE [9] is an architectural knowledge management portal that claims to implement best practices from knowledge management for improving architectural knowledge sharing. ADDSS [8] is a web-based tool developed for managing architectural decisions. ADDSS captures both architectures and decisions following an iterative process and simulates the way in which software architects build their architectures as a set of successive refinements. In spite of having architectural knowledge management capabilities of these tools they lack automation support which proves a big hurdle in their adaptation.

To develop AAKET, we have adopted tag-based/annotation-based knowledge identification and extraction approach. There are few other similar attempts as reported in [20], [13] but they have been implemented in the context of semantic web. Yoshikiyo et al. have adopted the same concept of tag-based design decisions extraction from emails and other documents [14]. However, their tool requires that the information first be stored in XML format rather than original document or email formats. This format restriction makes this tool less useful in practical situations.

Lee and Kruchten have proposed three approaches to capturing design decisions i.e. formal elicitation, lightweight top-down capture and lightweight bottom-up capture [17]. Their tool filters architectural information based on tags but they explore such information from source code of a project, which is optimistic supposition. Another part of their tool package supports saving emails which contain some architectural information but here they are not applying tag-based filtering mechanism on email contents.

GRIFFIN project [21] has also resulted in a few tools for managing architectural knowledge. However, these tools do not support knowledge extraction from emails, which we claim is a common way of sharing architectural knowledge. Nor do those tools focus on integrating the extracted knowledge with third-party knowledge repositories, whereas AAKET's architecture easily be modified to support different knowledge repositories such as ADDSS. Moreover, we also claim that AAKET is easier and simpler to use and we plan to gather more empirical evidence to support this claim.

7. Conclusion and Future Directions

Our main research object is to improve the process of software architecture by providing methods and tools for capturing and sharing AK. To achieve this objective, we have been developing various approaches and tool support for AKM. We also intend to reduce the time, resources and skill level required to effectively and efficiently capture

and manage AK. Our effort to provide a tool like AAKET has been motivated by our intention to help organizations and individuals to reduce the time and effort required for capturing AK in repository driven solutions being developed by AKM community including ourselves. The presented prototype is the outcome of our initial effort towards achieving this goal as it provides the proof that the concepts we are following are workable.

Our colleagues in AKM community and the participants of the described study have provided us with several ideas that need to be incorporated in the next version of AAKET before it is ready for an industrial trial or a large empirical study in academic environment. We are making progress on adding new features to AAKET to provide customizable and enhanced support for extracting and capturing AK with AAKET as identified by the participants of the described study. We anticipate being able to present more enhanced version of AAKET at the workshop to receive community feedback and critique. Some of the tasks for our immediate future work are outlined as follows:

Though the technology used for developing AAKET provides very good performance that has been observed during tool usage. However, we do not have any empirical data or solid evidence. We intend to conduct a detailed empirical evaluation of the performance of AAKET with large data set.

We support only MS-Word and MS-Outlook. It has also been seen that some decisions are communicated in MS-Excel format and in more informal way through chatting software. We intend to broaden the scope of automatic support for capturing and managing AK to these two areas as well. Moreover, we are also building new plug-ins for providing new tags (e.g., Rationale, scenario) and providing users with the facility of building the tags themselves. However, before adding these features to AAKET, we intend to conduct a field study in order to understand the current practices of using different communication and productivity software for codifying and sharing AK.

Wikis have become a very common source of information sharing. We are also planning to study the nature and amount of architectural information being shared using Wikis. Based on the findings of this study, we will consider the value and viability of adding support for knowledge extraction from Wikis.

Currently our "knowledge persistence" component only supports one remote repository i.e. PAKME. In near future, we will provide various APIs for supporting other repository driven AKM tools such as ADDSS [18] and others.

Acknowledgement: *This work is partially supported by Science Foundation Ireland under grant number 03/CE2/I303-1.*

8. References

[1] MPIAT. Mars Program Independent Assessment Team summary report, *Tech Report* 2000.

[2] M. Ali-Babar, R.d. Boer, T. Dingsoyr, and R. Farenhorst, Architectural Knowledge Management Strategies: Approaches in Research and Industry, *Proceedings of the 2nd Workshop on SHAring and Reusing architectural knowledge - Architecture, rationale, and Design Intent (SHARK/ADI 2007), Collocated with ICSE 2007.*, 2007.

[3] M. Ali-Babar, et al., Introducing Tool Support for Managing Architectural Knowledge: An Experience Report, *Proceedings of the 15th IEEE International Conference on Engineering Computer-Based Systems*, 2008.

[4] J. Anton and B. Jan, Software Architecture as a Set of Architectural Design Decisions, in Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture. 2005, IEEE Computer Society.

[5] T. Antony, J. Yan, and H. Jun, A rationale-based architecture model for design traceability and reasoning, *J. Syst. Softw.*, 2007. **80**(6): pp. 918-934.

[6] M.A. Babar and I. Gorton, A Tool for Managing Software Architecture Knowledge, in Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent. 2007, IEEE Computer Society.

[7] M.A. Babar, X. Wang, and I. Gorton, PAKME: A Tool for Capturing and Using Architecture Design Knowledge, *9th International Multitopic Conference, IEEE INMIC 2005*, 2005.

[8] R. Capilla, F. Nava, S. Pérez, and J.C. Dueñas, A web-based tool for managing architectural design decisions, *SIGSOFT Softw. Eng. Notes*, 2006. **31**(5): pp. 4.

[9] R. Farenhorst, R. Izaks, P. Lago, and H.v. Vliet, A Just-In-Time Architectural Knowledge Sharing Portal, in Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008) - Volume 00. 2008, IEEE Computer Society.

[10] R. Farenhorst, P. Lago, and H. van Vliet, Effective Tool Support for Architectural Knowledge Sharing, in Proceedings of the First European Conference on Software Architecture. 2007.

[11] A. Jansen, J.v.d. Ven, P. Avgeriou, and D.K. Hammer, Tool Support for Architectural Decisions, in Proceedings of the Sixth Working IEEE/IFIP Conference on Software Architecture. 2007, IEEE Computer Society.
[12] A. Jansen, T.d. Vries, P. Avgeriou, and M.v. Veelen, Sharing the Architectural Knowledge of Quantitative Analysis, in Proceedings of the Quality of Software-Architectures (QoSA 2008). 2008.
[13] J. Kahan and M.-R. Koivunen, Annotea: an open RDF infrastructure for shared Web annotations, in Proceedings of the 10th international conference on World Wide Web. 2001, ACM: Hong Kong, Hong Kong.
[14] Y. Kato, K. Hori, and K. Taketa, Capturing Design Rationale by Annotating E-mails, in In Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics. 2002, Int. Inst. Inf. & Syst: Orlando. pp. 278-82.
[15] P. Kruchten, Capturing architectural knowledge with PAKME, *(Personal communicatoin)*, 2008.
[16] P. Kruchten, P. Lago, and H. van Vliet, Building Up and Reasoning About Architectural Knowledge, in Quality of Software Architectures. 2006. pp. 43-58.
[17] L. Lee and P. Kruchten, Customizing the capture of software architectural design decisions, *Electrical and Computer Engineering, 2008. CCECE 2008. Canadian Conference on*, 2008.
[18] F. Nava, R. Capilla, and J. Dueñas, Processes for Creating and Exploiting Architectural Design Decisions with Tool Support, in Software Architecture. 2007. pp. 321-324.
[19] A. Tang, Y. Jin, and J. Han, A rationale-based architecture model for design traceability and reasoning, *J. Syst. Softw.*, 2007. **80**(6): pp. 918-934.
[20] M. Vargas-vera, et al., MnM: Ontology Driven Semi-automatic and Automatic Support for Semantic Markup, in Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web. 2002, Springer. pp. 213-221.
[21] H.v. Vliet and P. Lago. The GRIFFIN project. Last accessed on Available from: http://griffin.cs.vu.nl/.