Capturing and Using Software Architecture Knowledge for Architecture-Based Software Development

Muhammad Ali Babar, Ian Gorton, and Ross Jeffery
Empirical Software Engineering
National ICT Australia Ltd. and University of New South Wales, Australia
{malibaba, ian.gorton, ross.jeffery} @nicta.com.au

Abstract

Management of architecture knowledge is vital for improving an organization's architectural capabilities. Despite the recognition of the importance of capturing and reusing architecture knowledge, there is no suitable support mechanism. We have developed a conceptual framework to provide appropriate guidance and tool support for making tacit or informally described architecture knowledge explicit. This framework identifies different approaches to capturing implicit architecture knowledge. We discuss different usages of the captured knowledge to improve the effectiveness of architecture processes. This paper also presents a prototype of a web-based architecture knowledge management tool to support the storage and retrieval of the captured knowledge. The paper concludes with open issues that we plan to address in order to successfully transfer this support mechanism for capturing and using architecture knowledge to the industry.

Keywords: Software architecture, software quality, knowledge management, software reuse, process improvement, experience factory

1. Introduction

Software Architecture (SA) design and evaluation involves complex and knowledge intensive tasks [1, 2]. The complexity lies in the fact that tradeoffs need to be made to satisfy current and future requirements of a potentially large set of stakeholders, who may have competing vested interests in architectural decisions [3, 4]. The knowledge required to make suitable architectural choices is broad, complex, and evolving, and can be beyond the capabilities of any single architect.

Due to the recognition of the importance and far reaching influence of the architectural decisions, several approaches (such as Architecture Tradeoff Analysis Method (ATAM) [5], 4+1 views [6], Rationale Unified Process (RUP) [7] and architecture-based development [8]) have been developed to support architecting processes. While these approaches help manage complexity by using systematic approaches to reason about various design decisions, they provide very little guidance or support to capture and maintain the details on which design decisions are based, along with explanations of the use of certain types of design constructs (such as patterns, styles, tactics and so on). Such information represents architecture knowledge, which can be valuable throughout the software development lifecycle [9, 10].

Lack of a systematic approach to capture and use architecture knowledge may preclude organizations from growing their architecture capability and reusing architectural assets. Moreover, the knowledge concerning the domain analysis, architectural patterns used, design alternatives evaluated and design decisions made is implicitly embedded in the architecture and/or becomes tacit knowledge of the architect [2, 9, 11].

Apart from the architectural artifacts created during architecting activities, there are several other sources of architecture knowledge. These include architecture styles and patterns [12-14], design patterns [15], architecture and design tactics [13, 16]. While these sources are aimed at explicitly codifying different types of architecture knowledge, some vital pieces of knowledge are either omitted or informally described. For instance, many pattern documentation formats do not explicitly describe the "*forces*[1]" of a pattern. We have also found that each pattern's documentation informally describes the schemas of synergistic relationships among patterns, quality attributes and scenarios as a pattern's description include the scenarios that characterize the quality attributes supported or blocked by that pattern. This information can be captured as reusable artifacts in a format that provides architectural knowledge at a level of abstraction appropriate for the architecture design phase [17, 18].

Our research is aimed at improving the quality of architecting process. This is achieved by developing effective knowledge management (KM) structures to facilitate the management of implicit architecture knowledge generated during architecting process or informally described in sources such as [13-16]. We have been developing a support mechanism to facilitate the capture and use of architecture knowledge by using concepts from KM [19, 20], experience factories [21, 22], and pattern-mining [17, 23] paradigms.

[1] *The forces of a pattern describe the factors which can cause a problem if they interfere with one another. A pattern attempts to resolve clashes among those factors. Discussion of forces also captures tradeoffs in a pattern.*

This paper presents a conceptual framework for capturing implicit architecture knowledge as reusable artifacts and managing it with a knowledge repository. This makes such knowledge readily available to improve architecture-based software development process. The framework identifies various approaches to capture implicit and explicit design and process knowledge during architecting process, along with an approach to distil and document architecture knowledge from patterns. The novelty of the approach resides in its ability to incorporate all the components into an integrated approach, which has been incrementally implemented in a web-based tool.

The reminder of this paper is organized as follows. In Section 2 we describe the theoretical background and motivation that stimulated the research. Section 3 presents a conceptual framework for capturing architecture knowledge. Section 4 describes usages of the captured knowledge. A brief description of a prototype tool is given in Section 5 and Section 6 concludes the paper.

2. Theoretical Background and Motivation

2.1 Architecture-Based Development

Software architecture embodies some of the earliest design decisions, which are hard and expensive to change if found flawed during downstream development activities. Since the quality attributes (such as maintainability and reliability) of complex software systems largely depend on the overall SA [13], a systematic and integrated approach is required to address architectural issues throughout the software development lifecycle; such approach is called architecture-based development [8]. Figure 1 shows a high level process model of architecture-based development that consists of six steps, each having several activities and tasks. Later in the paper, we briefly describe what type of knowledge can be captured or used by each step.

Architectural requirements are those requirements that have broad cross-functional implications. Such requirements are usually elicited and specified using quality sensitive scenarios [13]. Architecture design is an iterative process, making incremental decisions to satisfy functional and architectural requirements. These decisions provide criteria to reason about the resulting architecture – this is called architecture analysis.

Architecture is documented in terms of views, each view addressing a different perspective of the architecture. Architecture design, documentation and analysis are iterative steps in the process [8]. Having designed and analyzed a suitable architecture, it is realized to create the system, and the architecture is maintained to ensure that the detailed design and implementation decision conform to the original architectural decisions and rationales.

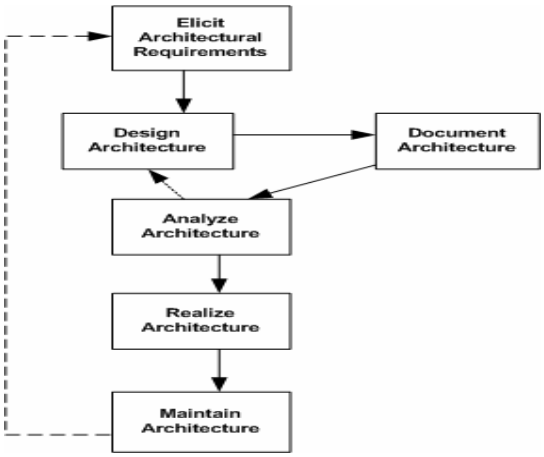

Figure 1: Architecture-Based Development Process model [8]

2.2 KM Problems in Architecting Processes

The architecture process aims to solve a mix of ill- and well-defined problems, which involve processing a significant amount of knowledge. Architects require topic knowledge (learned from text books and courses) and episodic knowledge (experience with the knowledge) [1]. One of the main problems in the architecture is the lack of access to knowledge underpinning the design processes and decisions [9, 24]. This type of knowledge involves things like the impact of certain middleware choices on communication mechanisms between different tiers, why an API is used instead of a wrapper, and who to contact to discuss the performance of different architectural choices.

Much of this knowledge is episodic and usually not documented [2]. The absence of a disciplined approach to capture and maintain architecture knowledge has many downstream consequences. These include:

- the evolution of the system becomes complex and cumbersome, resulting in violation of the fundamental design decisions
- inability to identify design errors
- inadequate clarification of arguments and information sharing about the design artifacts and process,

All these cause loss of substantial knowledge generated during the architecture process, thus depriving organizations of a valuable resource, loss of key personnel may mean loss of knowledge [2, 25, 26].

The SA community has developed several methods (such as ATAM [5], PASA [27]) to support a disciplined approach to architectural practices. Some of these do emphasize the need for knowledge management to improve reusability and grow organizational capabilities in the architecture domain. Except for [28], there is no approach that explicitly states what type of knowledge needs to be managed and how, when, where, or by whom. Also, none of the current approaches provides any conceptual framework

to design, develop and maintain an appropriate repository of architecture knowledge. Hence we posit that the lack of suitable techniques, tools, and guidance is why knowledge about design decisions is not captured and managed.

The software engineering community has been discovering and documenting architecture knowledge accumulated by experienced researchers and practitioners in the forms of patterns [14, 15]. However, we have found that the amount of information provided and the level of abstraction used may not be appropriate for the architecture stage – too much detail is counter-productive as expert designers usually follow breadth-first approach [1]. Moreover, we have found that the existing formats of pattern documentation are not appropriate for explicating the schemas of the relationships among scenario, quality attributes, and patterns in a way that makes this knowledge readily reusable. This results in little use/reuse of the architectural artifacts (such as scenarios, quality attributes and tactics) informally described in patterns' documentation [17, 18].

Like any other activity of software development, KM in architecting processes also suffers from other problems such as lack of motivation, resources, lackluster sponsorship by the management and others [29, 30]. However, these issues are not within the main focus of this paper.

2.3 Knowledge Management Building Blocks

The major objective of Knowledge Management (KM) is to improve business processes and practices by utilizing individual and organizational knowledge resources. These include skills, capabilities, experiences, routines, cultural norms, and technologies [19]. Software engineering processes need or generate both explicit and implicit knowledge. These are mutually complementary entities that interact with each other in various creative activities [31].

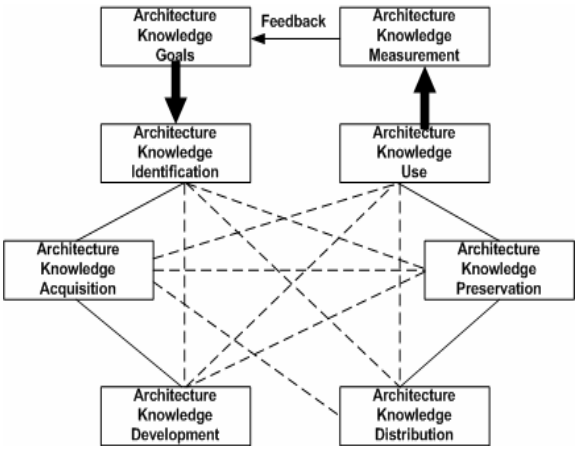

Figure 2: Building blocks of knowledge management (Modified for SA knowledge from [19]).

KM does not ignore the value or need to address other software development aspects, such as process and technology, nor does it seek to replace them. Instead, it works toward software process improvement by explicitly and systematically addressing the management of knowledge. This includes its acquisition, structuring, storage and effective maintenance [20]. There are two main strategies to manage knowledge:

1. codification: making tacit knowledge explicit
2. personalization: supporting knowledge sharing by describing who knows what.

Successful organizations apply both codification and personalization strategies: one of them in a primary and the other in a secondary role [32].

We posit that architecture knowledge management is a management task, which can be described using the knowledge management task model presented in [20]. This model (Figure 2) consists of two strategic and six operational knowledge management tasks, called the building blocks of KM. Each of the building blocks represents a particular task of managing knowledge and underpin an iterative model that presents an integrated approach to KM and ignoring one or more of the building blocks can interrupt the knowledge cycle [19]. For example, if contextual information about designing an artifact in a particular way is not preserved, it may disappear from organizational or individual memory, making reusability of that artifact difficult.

2.4 Experience Factory Organization

The Experience Factory Organization (EFO) provides a conceptual framework for building a systematic approach to accumulate and reuse domain specific knowledge [22]. The main objective of the EFO approach to improve the performance (in terms of cost, quality, and schedule) of software development projects by leveraging experience from previous projects [21]. The EFO framework takes into account the reality that accumulating and maintaining knowledge and experiences of software development are non-trivial tasks, which should not be left to individual projects. This is because it is difficult for a project team to devote resources to capture their experiences for reuse while deadlines are looming or quality and productivity have top priority.

The EFO addresses this issue by dividing the responsibilities of software development and experience accumulation into two organizational units:

1. Project Organization: uses packaged experience to deliver software products
2. the Experience Factory: supports software development by providing tailored experience [22].

Unlike the EFO, our approach treats the experience factory as a tool, called the Architecture Knowledge Repository (AKR), instead of a separate organizational unit. However, the AKR also has been divided into project knowledge (concrete) and corporate knowledge

(generic). Another requirement of reusability is an appropriate structure to enable tailoring and generalizing knowledge. We have addressed this issue by designing a set of templates to capture and represent both types (concrete and generic) of architecture knowledge [17].

3. Capturing Architecture Knowledge

This section presents a conceptual framework for capturing implicit knowledge. This framework provides a support mechanism to design, develop and populate a knowledge repository to improve architecture-based software development. The framework comprises planning, capturing, organization and evaluation, and storage of architecture knowledge (Figure 3).

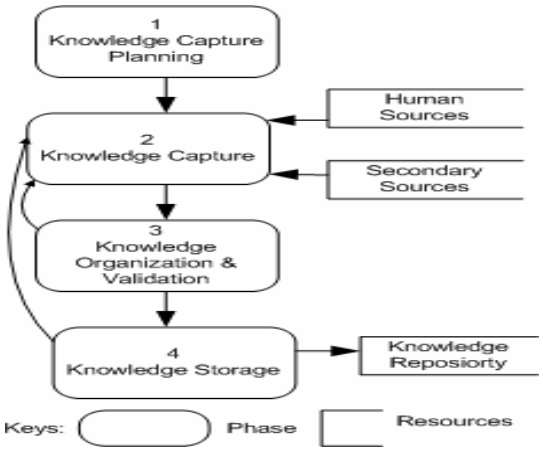

Figure 3: A conceptual framework for capturing architecture knowledge

The planning phase is aimed at understanding the knowledge of the domain, identifying the sources of the knowledge, and deciding about the techniques to be used. The main objective of knowledge capture phase is to acquire knowledge from human or secondary sources using the techniques described in section 3.1 and 3.2. The knowledge captured in this phase needs to be organized and evaluated (the objective of phase 3) before being placed in the AKR as a reusable artifact. There are different techniques (such as transcription, coding, summarization [33]) to organize knowledge depending on the knowledge capture source and methods. For example, we have developed different templates to organize knowledge extracted from patterns [18]. The organized knowledge is validated before being stored in knowledge repository. The cycle between phase 2-4 runs until most of the required implicit knowledge from the people or secondary sources has been extracted, organized and stored

3.1 Capturing knowledge from Human sources

One of the main sources of implicit architecture knowledge is people (e.g. architects, domain experts and project teams), who individually and collectively carry a large amount of "*know-how*" and "*community specific folklore*" about their domain and projects [2]. There are two main strategies to capture such implicit knowledge to populate a knowledge repository: 1) appoint a knowledge engineer to capture implicit knowledge from individuals or teams [2, 34] or 2) provide appropriate tool support so that knowledge can be encoded into the system as part of the knowledge creation process. The latter is called *contextualized* knowledge acquisition [35]. This strategy is similar to Electronic Process Guide (EPG) [36]. It is not the intent of this paper to recommend a particular strategy as each of them have been found useful in different contexts.

Table 1: Some Knowledge Acquisition Techniques

Individual Knowledge Acquisition Techniques	Team Knowledge Acquisition Techniques
Interviewing Questionnaire Observation Protocol analysis Repertory grid analysis	Brainstorming Architecture reviews Focus group interviews Delphi technique Group repertory grid analysis Group support systems

When applying the first strategy of knowledge acquisition, someone can use a variety of techniques derived from different disciplines such as expert systems, artificial intelligence, groupware systems and others. Table 1 presents some of the techniques that are useful to capture implicit knowledge. A succinct explanation of these techniques is provided in [37].

To implement the second strategy, a suitable environment is provided so that knowledge can be encode it is created [35]. We have developed a KM repository as a support mechanism for this strategy. However, an empty knowledge repository cannot motivate people to use it. Before exposing the potential users to a knowledge repository, it should be populated [38]. This can be done by capturing knowledge from experts using the above-mentioned techniques or from secondary sources such as patterns using the "*pattern-mining*" approach to populate the AKR.

3.2 Capturing knowledge from patterns

We have found that software patterns are a valuable source of architecturally significant constructs (such as scenarios and tactics) and relationships between them. These synergistic relationships should be captured and documented as reusable architecture knowledge to support and improve architecting activities [17, 18]. To facilitate the task of knowledge acquisition from patterns, we have developed:

- a process model to capture and structure architecture knowledge from patterns

- a set of guidelines to identify and capture the architectural information that can be captured as a reusable artifact from a pattern
- a set of templates to structure and document the extracted architecture knowledge

In the following, we describe the steps of the pattern-mining process (Figure 4).

The process consists of the following steps:

1. Select a software pattern to be explored for architectural information. This decision is usually influenced by a system's domain and the software engineer's experience.
2. Understand the pattern documentation format to identify the variations that exist among different patterns' description styles.
3. Explore different parts of the selected patterns to identify architectural information described in a pattern's documentation
4. Capture each type of information separately
5. Structure and document the extracted information using the provided template
6. Validate and refine documented information based on domain knowledge and experience of using different patterns.

Patterns are usually documented in a variation of the format used in [14, 15]. This requires the inclusion of problem, solution, and quality consequences parts. We have found that scenarios are mostly found in the problem and solution sections. Forces of a pattern may be described separately but usually forces can also be found in the problem and solution sections. The quality attributes are described in the quality consequence section, usually at the end of a pattern's description.

The extracted information must also be structured and documented in a format that creates a readily useable knowledge artifact. We have designed a set of templates to document different units of architecturally significant information (i.e. general scenarios, quality attributes, tactics, usage examples and so on) as an artifact of architecture knowledge.

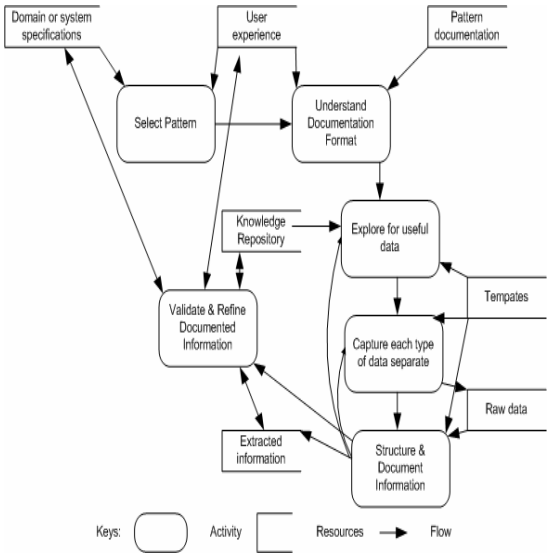

Figure 4: A process model of mining patterns for architecture knowledge from patterns

Table 2 presents one of these templates. The template presents different pieces of a pattern's description in a succinct format at an abstraction level suitable for SA design and evaluation stages, where abstract scenarios are used to characterize required quality attributes and suitable patterns are chosen based on their support for the required quality attributes. This template contains the knowledge extracted from *Business Delegate* J2EE pattern by following the *pattern-mining* process.

Table 2: Abstract architecture knowledge extracted from J2EE Business Delegate pattern

Pattern Name: *Business Delegate*		**Pattern Type:** *Design pattern*	
Brief description		*This pattern reduces coupling between tiers by providing an entry point for accessing the services another tier. It also supports results caching to improve performance…*	
Context		*A client may be exposed to the complexity of dealing with the distributed components…*	
Problem description		*Presentation-tier components interact directly with business services. Such a direct interaction makes the clients vulnerable to any changes in the business services…*	
Suggested solution		*Reduce coupling between presentation-tier clients and business services. The Business Delegate hides the underlying implementation details of the business service...*	
Forces		*Presentation-tier clients require access to business service.* *It is desirable to minimize coupling to hide implementation details from clients.*	
Available tactics		*Delegate Proxy and Delegate Adapter*	
Affected Attributes		**Positively**	**Negatively**
		Reduce coupling, manageability. performance	*Introduce new layer, increased complexity*
General scenarios	S1	*Presentation-tier components shall not be exposed to the implementation details of the business services they use.*	
	S2	*System shall provide a caching mechanism to improve response to business service request.*	
	S3	*Changes in the business services implementation shall not require corresponding changes in their clients residing in other tier.*	
Usage examples		*E-commerce portals, online content providers, sports websites.*	

1550-6002/05 $20.00 © 2005 IEEE

The template makes the relationships among scenarios, quality attributes, and patterns explicit. Moreover, it also captures one of the most important parts of a pattern description, namely the *forces*. The forces of a pattern are usually described implicitly in most of the pattern documentation styles. Recently, there are some efforts to pay more attention to the forces of a pattern [39, 40]. The abstract knowledge captured in template 1 can be concretized for a specific project. For example, general scenarios extracted from patterns are concretized to specify quality attributes. It also increases confidence in an architecture's capability of satisfying certain concrete scenarios if they are instances of a general scenarios extracted from a pattern used in that architecture [17].

4. Using Architecture Knowledge

In this section, we describe a few ways of using the architectural knowledge captured and organized using the techniques described in section 3 and stored in the AKR. The generic architectural knowledge captured from patterns, experts, past project or other sources can be reused in new project by instantiating general scenarios, architectural decisions, and modifying patterns according the context of a project. Project-specific knowledge is used not only to support various architecture activities, but also can be transformed into reusable artifacts. For instance, general scenarios stored in the repository can be used to instigate thinking while eliciting quality requirements (Figure 1) or these scenarios can be used to generate concrete scenarios.

During the design stage, generic knowledge should help architects to identify suitable patterns by comparing the scenarios and quality attributes supported by different patterns with the ones required by the stakeholders. Moreover, architects can also evaluate the suitability of generic architecture decisions suggested for a particular context and they can contact the contributor of a particular architecture decision for further explanation. SA evaluation activities can also be improved by using both generic and project-specific knowledge. Generic knowledge helps improve the scenario development task, select suitable reasoning frameworks and increase confidence in the capabilities of architecture to satisfy particular quality sensitive scenarios as a result of using certain patterns [18, 23].

Project-specific knowledge helps designers, developers and maintainers to better understand the architectural decisions, their constraints and reasoning behind it. Moreover, the availability of the reasoning behind the architectural decisions helps architects to explain architectural choices and how they satisfy business goals [11]. Such knowledge is also valuable during the architecture realization and maintenance stages (Figure 1) of architecture-based development processes. We have designed an empirical research program to assess different uses of the knowledge captured from pattern and preliminary results are very encouraging [17].

5. PAKME - A Prototype of Architecture Knowledge Management System

The Process-based Architecture Knowledge Management Environment (PAKME) is a prototype web-based system to provide knowledge management support for improving architecting activities. The PAKME has been built on top of an open source groupware platform, Hipergate [41]. This provides various collaborative features including contact management, project management, online collaboration tools and others. We have modified the data model of the Hipergate to add the features required to capture, manage, and retrieve architecture knowledge; the AKR database currently consists of 25 tables.

The knowledge repository is logically divided into *knowledge-based artifacts, generic knowledge, and project-based artifacts*. The generic knowledge is accumulated by using the implicit knowledge capture techniques described in this paper. So far we have populated the ARK by distilling architecture knowledge from several J2EE [42] patterns, architecture patterns [14], and the BCS case study described in [13]. Project-based architecture knowledge consists of the artifacts either instantiated from generic knowledge or newly created during various architecture activities.

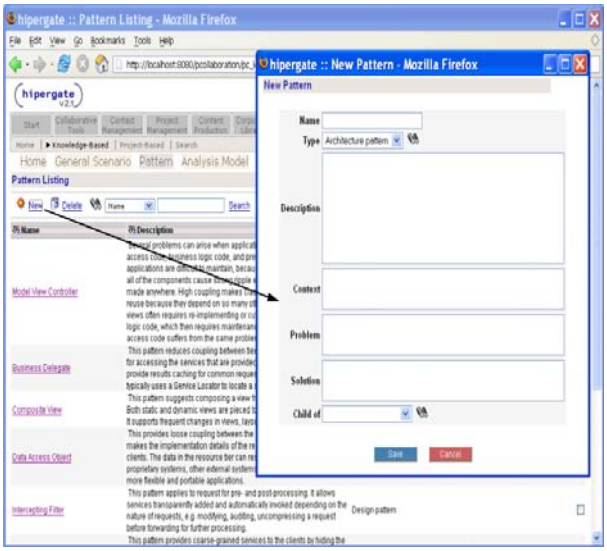

Figure 5: A form for entering a new pattern in the AKR

Currently, the PAKME consists of four components; knowledge acquisition, knowledge maintenance, knowledge retrieval, and knowledge presentation. The knowledge acquisition component provides various forms and editing tools to enter new generic or project-specific knowledge in the repository. The forms are based on the templates (e.g. Table 2) developed to organize knowledge. Figure 5 shows a form for entering a new pattern in the AKR. While entering a new artifact, an end user can view the existing artifacts in the

background as shown in Figure 5. For example, if a user's search fails to retrieve a particular pattern, the user may decide to enter that pattern in the repository. The knowledge maintenance component provides various features to modify, delete and instantiate different architectural artifacts. It also includes repository administration functions.

The retrieval component supports both basic and advanced searches to find and retrieve the desired pieces of knowledge. Moreover, a user can traverse to different related artifacts by navigating through the knowledge space based on the initial search results (Figure 6).

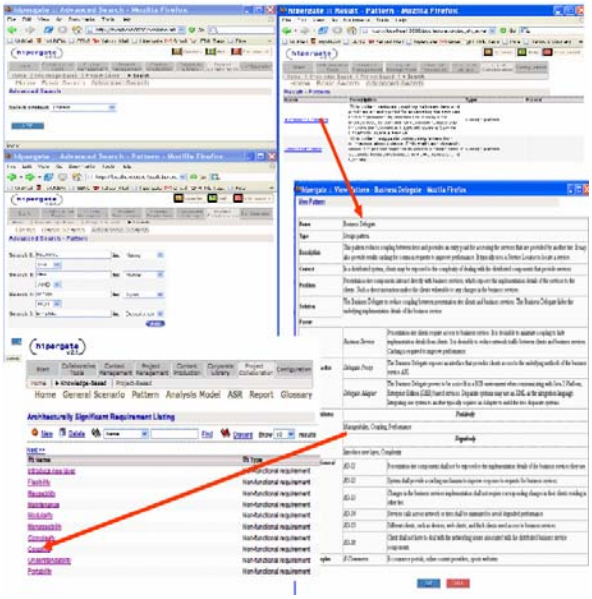

Figure 6: Various screens showing the results of search and navigation based retrieval provided by the AKR.

The knowledge presentation component supports generating different views of the architecture knowledge residing in the AKR. For example, it presents utility (Figure 7) trees and result trees based on the results of architecture evaluation sessions using ATAM [13]. To summarize, the two main objectives of the PAKME are:

- To provide a support mechanism for knowledge capture, manage, and retrieval to improve the quality of architecting activities.
- To act as a knowledge source for those who need rapid access to experience-based design decisions to assist in making new decisions or discovering the rationale for past decisions.

6. Conclusion and Future work

Our research is aimed at improving the effectiveness of SA processes by providing suitable support mechanisms. Current approaches are deficient in providing the required design knowledge or managing the knowledge generated. This leads to a lack of use of existing SA knowledge as it is not available in a readily usable format at an appropriate level of abstraction. Moreover, implicit knowledge is not normally captured to make it available for decision support.

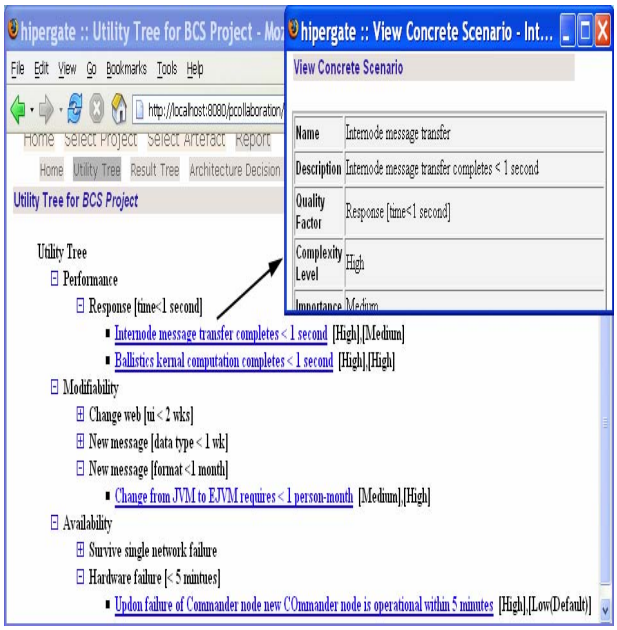

Figure 7: A utility tree of concrete scenarios.

This paper emphasizes the importance of capturing and using implicit SA design knowledge to improve architecture activities. We present a framework for capturing implicit knowledge using various knowledge acquisition and pattern-mining techniques and structuring and storing that knowledge in a knowledge repository developed to support the framework.

Future work includes enhancement of the tool with case-based approaches [35] and incremental refinement of search queries based on the results of the basic search. We are particularly keen to test the pattern-mining process and tool in industrial settings, so that their applicability and scalability can be thoroughly assessed. The preliminary results of our assessment of the pattern-mining process and the usefulness of the extracted knowledge are very encouraging [17]. These give us confidence in the utility of our approach.

Acknowledgement – *Several undergraduate students helped us build the tool. National ICT Australia is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.*

7. References

[1] Robillard, P.N., "The role of knolwedge in software development," *Communications of the ACM*, 1991. **42**(1): pp. 87-92.

[2] Terveen, L.G., et al., "Living Design Memory: Framework, Implementation, Lessons Learned," *Human-Computer Interaction*, 1995. **10**(1): pp. 1-37.
[3] Gorton, I. and J. Haack, "Architecting in the Face of Uncertainty: An Experience Report," *Proc. International Conference on Software Engineering*. 2004.
[4] Al-Naeem, T., et al., "A Quality-Driven Systematic Approach for Architecting Distributed Software Applications," *Accepted in 27th Int'l. Conf. of Software Eng.* 2005.
[5] Clements, P., et al., "Evaluating Software Architectures: Methods and Case Studies". Addison-Wesley, 2002.
[6] Kruchten, P.B., "The 4+1 View Model of architecture," *Software, IEEE*, 1995. **12**(6): pp. 42-50.
[7] Kruchten, P., "The Rational Unified Process: An Introduction", 2nd ed. Addison-Wesley, 2000.
[8] Bass, L. and R. Kazman, "Architecture-Based Development," Tech. Report CMU/SEI-99-TR-007, SEI, Carnegie Mellon University, Pittsburgh, 1999
[9] Bosch, J., "Software Architecture: The Next Step," *European Workshop on Software Architecture*. 2004.
[10] Dutoit, A.H. and B. Paech, "Rationale Management in Software Engineering", in *Handbook of Software Engineering and Knowledge Engineering*, S.K. Chang, Editor. World Scientific Publishing, 2001, Singapore.
[11] Tyree, J. and A. Akerman, "Architecture Decisions: Demystifying Architecture," *IEEE Software*, 2005. **22**(2): pp. 19-27.
[12] Shaw, M. and D. Garlan, "Software Architecture: Perspectives on an Emerging Discipline". Prentice Hall, Upper Saddle River, NJ, 1996.
[13] Bass, L., et al., "Software Architecture in Practice", 2 ed. Addison-Wesley, 2003.
[14] Buschmann, F., et al., "Pattern-Oriented Software Architecture: A System of Patterns". John Wiley & Sons, 1996.
[15] Gamma, E., et al., "Design Patterns-Elements of Reusable Object-Oriented Software". Addison-Wesley, Reading, MA, 1995.
[16] Bachmann, F., et al., "Deriving Architectural Tactics: A Step toward Methodical Architectural Design," Tech. Report CMU/SEI-2003-TR-004, SEI, Carnegie Mellon University, USA, 2003
[17] Ali-Babar, M., et al., "Mining Patterns for Improving Architecting Activities - A Research Program and Preliminary Assessment," *Proc. of 9th Int'l. conf. on Empirical Assessment in Software Engineering*. 2005.
[18] Ali-Babar, M., "Scenarios, Quality Attributes, and Patterns: Capturing and Using their Synergistic Relationships for Product Line Architectures," *Proc. of the Int,l. Workshop on Adopting Product Line Software Engineering*. 2004.
[19] Probst, G.J.B., "Practical Knowledge Management: A Model That Works", http://know.unige.ch/publications/Prismartikel.PDF. Last accessed on 14th March, 2005
[20] Rus, I. and M. Lindvall, "Knowledge Management in Software Engineering," *IEEE Software*, 2002. **19**(3): pp. 26-38.
[21] Basili, V.R. and G. Caldiera, "Improving Software Quality Reusing Knowledge and Experience," *Sloan Management Review*, 1995. **37**(1): pp. 55-64.
[22] Basili, V.R., et al., "The Experience Factory", in *Encyclopedia of Software Engineering*, J.J. Marciniak, Editor. John Wiley & Sons, 2001.
[23] Zhu, L., et al., "Mining Patterns to Support Software Architecture Evaluation," *Proc. of the 4th Working IEEE/IFIP Conference on Software Architecture*. 2004.
[24] Ali-Babar, M., et al., "A Framework for Supporting Architecture Knowledge and Rationale Management", in *Rationale Management in Software Engineering*, A.H. Dutoit, et al., Editors. Submitted for review, 2005.
[25] Gruber, T.R. and D.M. Russell, "Design Knowledge and Design Rationale: A Framework for Representing, Capture, and Use," Tech. Report KSL 90-45, Knowledge Systems Laboratory, Standford University, California, USA, 1991
[26] Jarczyk, A.P.J., et al., "Design Rationale for Software Engineering: A Survey," *Proc. 25th Hawaii Int'l. Conf. on System Sciences*. 1992.
[27] Williams, L.G. and C.U. Smith, "PASA: An Architectural Approach to Fixing Software Performance Problems," *Proc. of Int'l. Conference of the Computer Measurement Group*. 2002.
[28] Clements, P., et al., "Documenting Software Architectures: Views and Beyond". Addison-Wesley, 2002.
[29] Davenport, T.H. and L. Prusak, "Working Knowledge". Harvard Business School Press, Boston, Massachusetts, 1998.
[30] Tiwana, A., "The Knowledge Management Toolkit: Orchestrating IT, Strategy, and Knowledge Platforms", 2nd ed. Prentice-Hall, 2002.
[31] Nonaka, I. and H. Takeuchi, "The Knowledge-Creating Company". Oxford University Press, 1995.
[32] Hansen, M.T., et al., "What's your strategy for managing knowledge?," *Harvard Business Review*, March-April 1999: pp. 106-116.
[33] Land, L.P.W., et al., "Capturing Implicit Software Engineering Knowledge," *Proc. 13th Australian Software Engineering Conference*. 2001.
[34] Skuce, B., "Knowledge management in software design: a tool and a trial," *Software Engineering Journal*, Sept. 995: pp. 183-193.
[35] Henninger, S., "Tool Support for Experience-Based Software Development Methologies," *Advances in Computers*, 2003. **59**: pp. 29-82.
[36] Scott, L., et al., "Understanding the use of an electronic process guide," *Journal of Information and Software Technology*, 2002. **44**(10): pp. 601-616.
[37] Liou, Y.I., "Collaborative Knowledge Acquisition," *Expert Systems With Applications*, 1992. **5**(1-2): pp. 1-13.
[38] Schneider, K. and T. Schwinn, "Maturing Experience Base Concepts at Daimler Chrysler," *Software Process Improvement and Practice*, 2001. **6**(2): pp. 85-96.
[39] Gross, D. and E. Yu, "From Non-Functional Requirements to Design through Patterns," *Proc. of the 6th Int'l Workshop on Requirements Engineering Foundation for Software Quality*. 2000.
[40] John, B.E., et al., "Bringing Usability Concerns to the Design of Software Architecture," *Proc. of the 9th IFIP Working Conference on Engineering for Human-Computer Interaction*. 2004.
[41] "Hipergate - Open Source CRM and Groupware", http://www.hipergate.com. Last accessed on 16th April, 2005
[42] Alur, D., et al., "Core J2EE Patterns: Best Practices and Design Strategies", 2nd ed. Sun Microsystem Press, 2003.