

ULRR

A tool for managing software architecture knowledge

Item Type	Meetings and Proceedings
Authors	Ali Babar, Muhammad;Gorton, Ian
Citation	Proceedings of 2nd Workshop on Sharing and Reusing Architecture Knowledge - Architecture, Rationale, and Design Intent (SHARK/ADI 2007), collocated with the 29th International Conference on Software Engineering (ICSE 2007;
Publisher	IEEE Computer Society
Download date	2026-06-13 09:43:45
Item License	https://creativecommons.org/licenses/by-nc-sa/1.0/
Link to Item	https://hdl.handle.net/10344/2393

A Tool for Managing Software Architecture Knowledge

Muhammad Ali Babar
Lero, University of Limerick, Ireland
Muhammad.Alibabar@ul.ie

Ian Gorton
Pacific Northwest National Laboratory
ian.gorton@pnl.gov

Abstract

This paper describes a tool for managing architectural knowledge and rationale. The tool has been developed to support a framework for capturing and using architectural knowledge to improve the architecture process. This paper describes the main architectural components and features of the tool. The paper also provides examples of using the tool for supporting well-known architecture design and analysis methods.

1. Introduction

Although significant progress has been made to support the architecture process over the last decade, little effort has been spent on developing techniques and tools for effectively managing knowledge pertaining to software architecture. Architecture knowledge can mainly be classified in two categories, namely contextual and technical. The former is called design rationale (DR) [1, 2] and provides the answers to questions about a certain design choice or the process followed to make that choice [3, 4]. If it is not captured, knowledge concerning the domain analysis, patterns used, design options evaluated, and decisions made is lost, and so is unavailable to support subsequent decisions [5-7]. The other type of knowledge is technical (such as patterns, styles, tactics, and analysis models) [8]. Such knowledge is required to design and evaluate architectures.

Recently, various researchers [9, 10] have proposed different ways to capturing contextual knowledge underpinning design decisions. Essential requirement of all these approaches is to describe architecture in terms of design decisions and DR surrounding them. However, design decisions and their rationale are not rigorously documented. One of the main reasons for this is lack of suitable methodological and tool support [11].

We have developed a framework for managing architecture knowledge (technical and contextual). This framework consists of techniques for capturing design decisions and contextual information, an approach to distill and document architectural knowledge from patterns, and a data model to characterize architectural constructs, their attributes and relationships [8, 12].

In order to support this framework, we have developed a web-based tool called PAKME (Process-centric Architecture Knowledge Management Environment). This paper describes various aspects of PAKME by providing examples of using it to support

methods of architecture design and analysis reported in [13, 14]. PAKME is also designed 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. Thus, PAKME serves as a repository of an organisation's architecture knowledge analogous to an engineers' handbooks, which consolidate knowledge about best practices in a certain domain [15].

2. Knowledge management tool support

PAKME is a web-based architecture knowledge management tool that is aimed at providing knowledge management support for the software architecture process. It has been built on top of an open source groupware platform, Hipergate [16]. This platform provides various collaborative features including contact management, project management, online collaboration tools and others that can be exploited to build a groupware support environment. This environment incorporates architecture knowledge management features for geographically distributed stakeholders involved in the software architecture process.

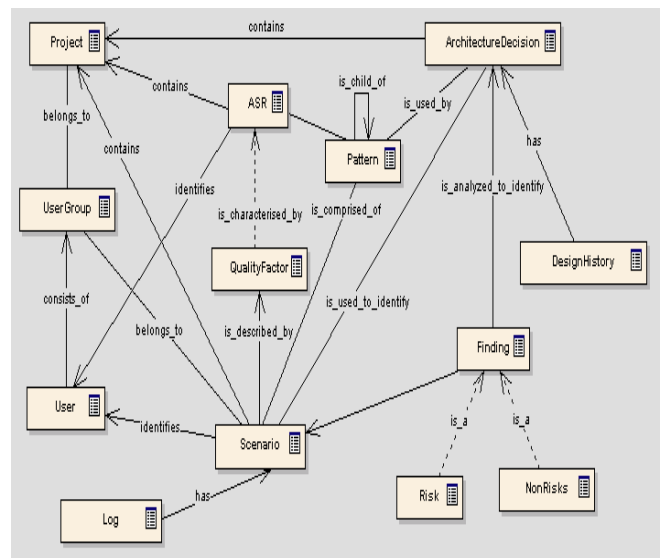


Figure 1: A partial data model characterising architectural artefacts captured in the software architecture knowledge base.

Like the modelers of measurement data [17], we also believe that a data model is one of the earliest artefacts

needed for the development of an automated system for storing and accessing the data that underpins architecture knowledge. Figure 1 presents a partial data model that identifies the main architectural constructs and their relationships¹. The data model divides architecture knowledge into organisational (generic), and project-specific (concrete). Access to a repository of generic knowledge enables designers to use accumulated “wisdom” from different projects when devising or analysing architectural decisions. The project-specific repository captures and consolidates knowledge and rationale specific to a project such as design history, analysis findings, and architectural views for stakeholders.

PAKME’s data model has been implemented by modifying the data model of Hipergate. We have also modified the presentation and business logic tiers of Hipergate in order to add the features required to manage architectural knowledge. Currently PAKME consists of four components:

User Interface – The only way to interact with the system is through Hipergate’s interface, which has been implemented using Java Server Pages (JSP) and HTML technologies. The user interface component provides various forms and editing tools to enter new organisational or project-specific knowledge in the repository. The knowledge acquisition forms are designed based on the templates proposed in [12].

Knowledge management – This component provides services to store, retrieve, and update artifacts that make up architectural knowledge. It has been mentioned that the knowledge base is logically divided into generic and concrete knowledge. This component also provides services to instantiate generic artifacts into concrete artifacts. This component uses the services of the data management component to manage artifacts.

Search – This component helps users search the desired artefacts. There are three types of search functions: keyword-based search, advanced search, and navigation-based search. Keyword-based search facility explores the repository for a desired artefact utilizing the key words that are attached as meta-data to each artefact. Advanced search is based on a combination of logical operators (i.e. And, Or and Not), while navigation-based search means searching the artefacts based on the results of the two main search functions. Navigational search is provided by presenting the retrieved artefacts as hyperlinks, which can be clicked to retrieve detailed information about them and other related artefacts.

Reporting – This component provides the services for representing architectural knowledge to explicate the relationships that exist between different architectural artefacts or to show their positive or negative effects on each other. For example, a tactic-benefit matrix shows which scenarios can be achieved by using which patterns through the tactics applied by those patterns. Furthermore, this component can generate various types of reports based on architecture evaluation results.

¹ The complete data model cannot be reported at this stage because of intellectual property issues.

Repository Management – This component provides all the services to store, maintain, and retrieve data from a persistent data source, which is implemented with PostgreSQL 8.0. The data management logic uses Postgres’s scripting language.

3. Managing architectural knowledge

Most of the approaches to managing knowledge can broadly be categorized into codification and personalization [18]. Codification concentrates on identifying, eliciting and storing knowledge as information in repositories, which are expected to support high-quality, reliable, and rapid reuse of knowledge. Personalization resorts to fostering interaction among knowledge workers for explicating and sharing knowledge. Although this paper focuses on the features of PAKME that support codification, this tool also supports personalization as it not only provides access to architectural knowledge but also identifies the source of knowledge. That means it can also support a hybrid strategy for managing knowledge [19]. Here we briefly discuss the four main services of PAKME:

- The knowledge acquisition service provides various forms and editing tools to enter new generic or project specific knowledge into the repository. The knowledge capture forms are based on various templates that we have designed to help maintain consistency during knowledge elicitation and structuring processes.
- The knowledge maintenance service provides different functions to modify, delete and instantiate the artifacts stored in the knowledge repository. Moreover, this service also implements the constraints on the modifications of different artifacts based on the requirements of a particular domain.
- The knowledge retrieval service helps a user to locate and retrieve desired artifacts along with the information about the artifacts associated with them. PAKME provides three types of search mechanisms. A basics search can be performed within a single artifact based on the values of its attributes or keywords. An advanced search string is built using a combination of logical operators within a single or multiple artifacts. Navigational search is supported by presenting the retrieved artifacts and their relationships with other artifacts as hyperlinks.
- The knowledge presentation service presents knowledge in a structured manner at a suitable abstraction level by using templates (such as provided in [12]) and representation mechanisms like utility and results trees described in [20].

These services not only satisfy the requirements identified by us to provide knowledge management support for methods like [13, 14], but also support many of the use cases proposed in [10]. {TO DO fix formatting]

Name	Description	Source	Date Entered	Logs	
SEC-1	QVS shall accept online payments for the services that means the transactions between the QVS and financial institutes must be protected.	User-Defined	Sun 24 Dec 2006 19:47		<input type="checkbox"/>
SEC-2	QVS provides secured storage to customers' credit details and other information.	User-Defined	Sun 24 Dec 2006 19:47		<input type="checkbox"/>
SEC-3	QVS shall be able to identify different users and verify their access privileges according to their memberships of different user groups.	User-Defined	Sun 24 Dec 2006 19:49		<input type="checkbox"/>
SEC-4	QVS shall be able to detect and prevent Denial Of Service (DOS) attacks. The system shall be able to run reliably most of the time.	User-Defined	Sun 24 Dec 2006 19:49		<input type="checkbox"/>
SEC-5	QVS is an evolving system that shall be easily modifiable to introduce changes in the security policy and other security checks.	User-Defined	Sun 24 Dec 2006 19:50		<input type="checkbox"/>
Aggregating data	A user may want to perform one or more actions on more than one object. Systems should allow users to select and act upon arbitrary combinations of data.	User-Defined	Sun 24 Dec 2006 23:35		<input type="checkbox"/>

Figure 2: General scenarios captured by PAKME's repository.

3.1 Capturing and presenting knowledge

There are two main strategies to elicit and codify knowledge:

- 1) Appoint a knowledge engineer to elicit and codify knowledge from individuals or teams [21, 22];
- 2) Provide a tool to encode the knowledge into the system as part of the knowledge creation process.

The latter is called contextualised knowledge acquisition [23], and each strategy has its strengths and weaknesses. To take the advantage of the strengths of both strategies, PAKME helps elicit and codify architecture knowledge using either of these strategies. We have been using PAKME by embedding it into knowledge creation processes. Its repository has been populated by capturing knowledge from several J2EE [24] patterns and architecture patterns [25], case studies described in [20, 26] and design primitives [27].

Figure 3: The interface to capture a general scenario.

PAKME provides several forms based on different templates to help users elicit and structure knowledge before storing it into the repository. Templates are aimed at keeping the process consistent across users [14]. Figure 3 shows a form for capturing a general scenario, which can be elicited from a stakeholder or extracted from a pattern. Each scenario can have several attributes attached to it including source documents,

revision history, and a set of keywords. PAKME's repository contains hundreds of general scenarios (Figure 2 shows some of them).

Figure 4: Template to capture and present patterns

Figure 4 shows a template for capturing and presenting patterns irrespective of the level of granularity (i.e., architecture, design, or framework-based). A pattern may be composed of other patterns and each pattern may have several tactics attached to it. To support reusability at the design decision level, PAKME's repository contains design options, which are design decisions that can be considered and/or evaluated to satisfy one or more functional or non-functional requirements. For example, Java RMI or publish-scribe design options can be used for event notification purposes. Each design option is composed of one of more architectural and/or design patterns and each of them is composed of one or more tactics. For example, the publish-subscribe design option applies the publish-on-demand design pattern.

PAKME captures design options as contextualized cases from literature or previous projects. A design option case consists of problem and solution statements, patterns and tactics used, rationale, and related design options. Rationale for each design option are captured in a separate template, which is designed based on practitioners' opinions about rationale reported in [11] and templates proposed in [5, 28]. Figure 4 shows a partial description of a design option. By capturing design options as cases, PAKME enables architects to follow a case-based approach and supports human-intensive case-based reasoning [29].

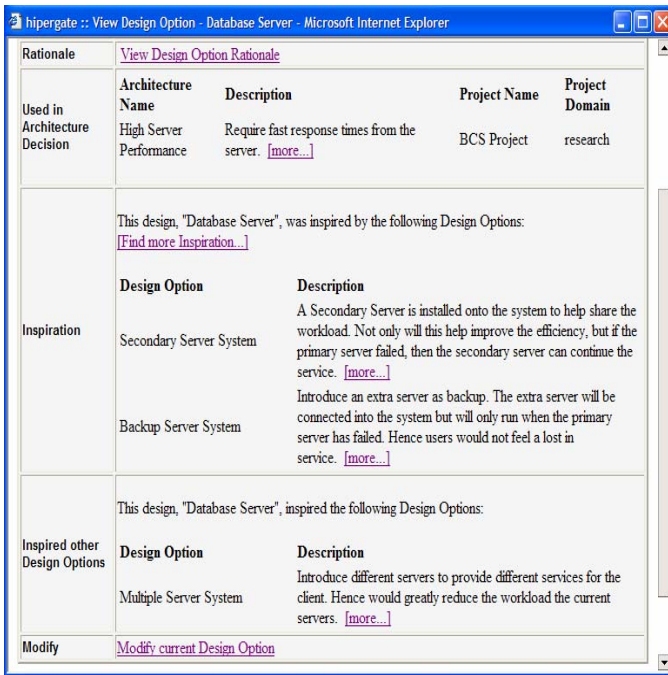


Figure 5: A partial view of a design option case.

Recently, there has been an increased emphasis on describing software architecture as a set of design decisions [6, 30]. Kruchten et al. have proposed a taxonomy of architectural decisions, their properties, and relationships among them [10]. Figure 6 shows that PAKME can capture many of the attributes and relationships of architectural decisions as described in [10] using templates proposed in [5].

In PAKME, architectural decision can be described at different levels of granularity as an architectural decision is a selected design option, which can be composed of an architectural pattern, a design pattern or a design tactic. Like a design option, each architecture decision also captures rationale using a template. The rationale describes the reasons for an architecture decision, justification for it, tradeoffs made, and argumentation leading to the design decision. Hence, PAKME captures rationale for design options as well as for architectural design decisions, which are made by selecting one or more suitable design options from a set of considered/assessed design options.

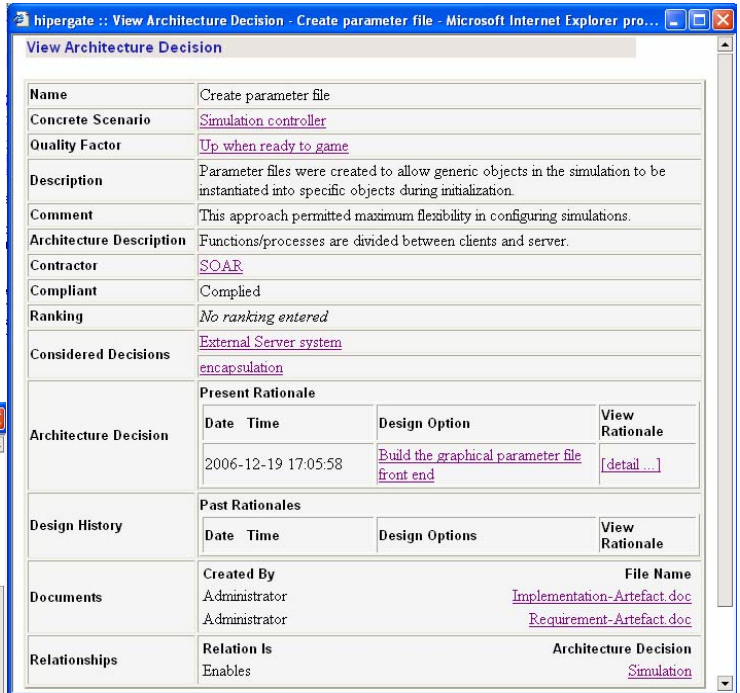


Figure 7: An architecture decision captured in PAKME

Moreover, traceability is also provided. Each architectural design decision describes the design options considered but rejected, concrete scenarios to be satisfied, and a model of architectural decision attached as design artifacts (shown in Figure 6). Revisions to design decisions and reasons are logged for later review. Design decisions are time stamped and annotated with the decision maker's details, which can be used to seek explanation for that design decision. Hence, we believe that PAKME supports the description of an architecture design decision in ways suggested in [5, 30] with the attributes and relationships proposed in [10]. Figure 6 shows that a user can establish several types of relationships among architecture design decisions.

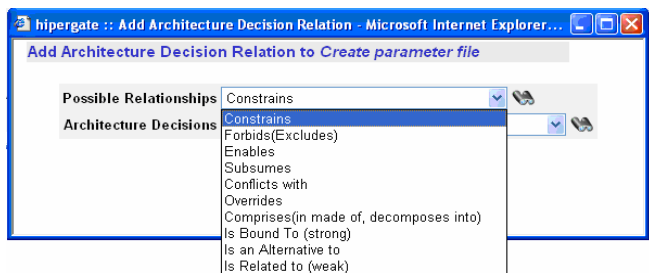


Figure 6: Types of relationships that can be established.

3.2 Supporting knowledge use/reuse

This section describes various ways in which PAKME facilitates architecture knowledge use/reuse. Let us first consider how PAKME supports the reuse of design options in making architecture decisions. Figure 5 shows that there is a four step process for reusing design options, which are captured as contextualized cases.

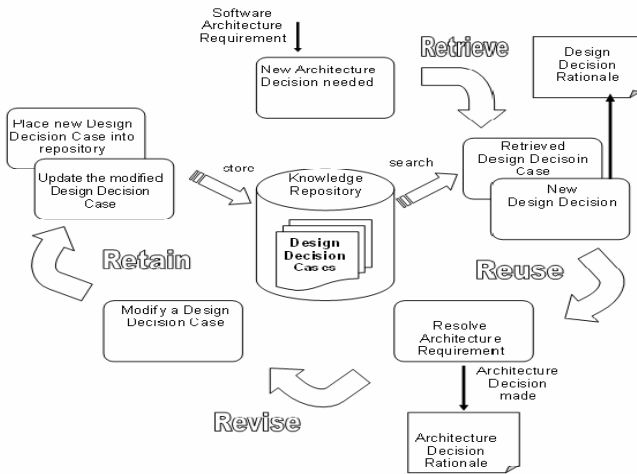


Figure 7: Process model of reusing design options

The process starts when a user has a new requirement that needs architectural support. This requirement would characterise a quality goal and would have been specified using concrete scenario. In order to satisfy that requirement, an architect needs to make a new architecture design decision. To address that requirement, the architect would then have two options:

- Search and retrieve a previous design option from the knowledge repository;
- Create a new design option to solve the given problem. For a new design option, the architect would also need to document the rationale.

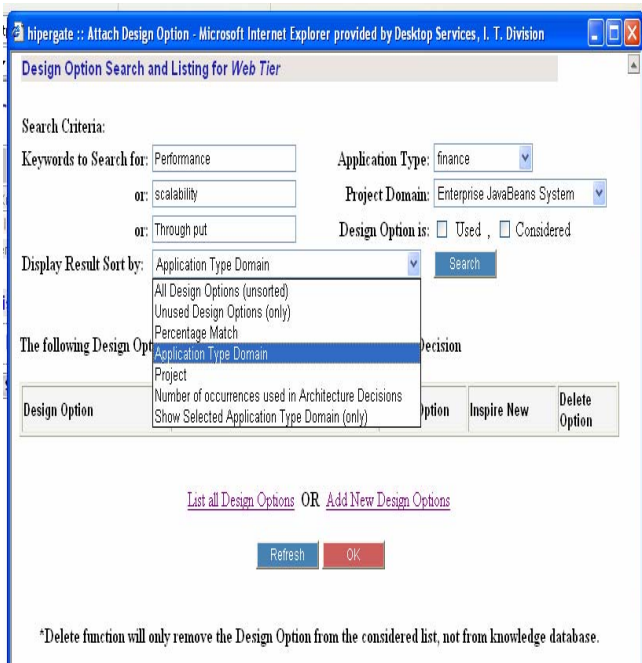


Figure 8: PAKME's interface for searching design option cases.

If the architect decides to search through the knowledge repository for cases of design options, they

can perform a search to retrieve a list of design options. Figure 8 shows that a user can build a complex search string based on various attributes. After reviewing the retrieved list of design options, the architect can either reuse an existing design option in its original form or modify it according to the current context. Figure 9 shows that a retrieved design option can be used by attaching it to an architecture design decision. If a design option is modified, it is considered a new design option but it is linked with the original design option for traceability. This new design option can be chosen as an architecture design decision through an attachment.

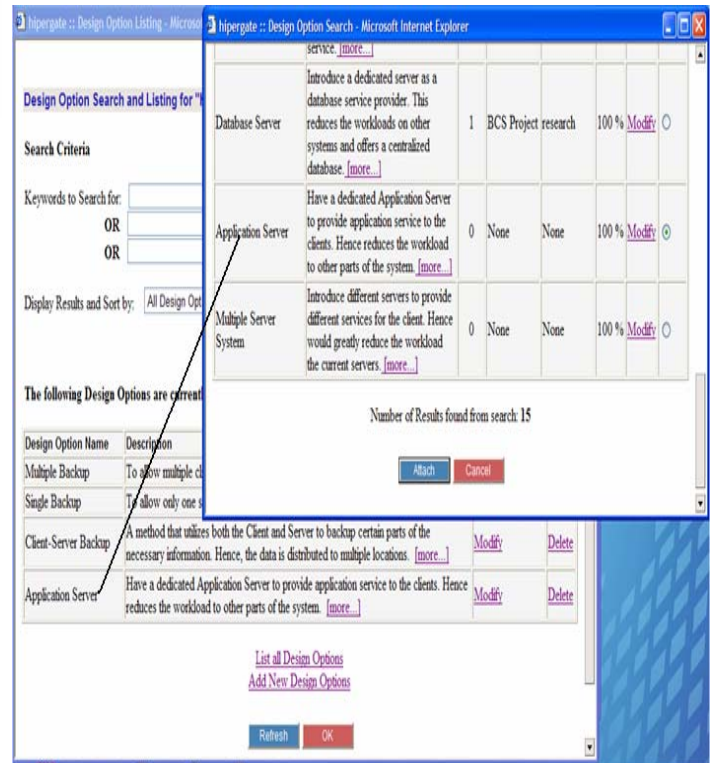


Figure 9: Attaching a retrieved design option to an architecture design.

To demonstrate the other ways of reusing architecture knowledge with PAKME, let us consider that an architect needs to design a suitable architecture for a new application. The architect is likely to make architectural decisions using a common process, namely understanding the problem, identifying potential alternatives, and assessing their viability.

There are several ways PAKME can support this process. The architect can search the repository for architectural artifacts that can be reused. For example, they can use a particular quality attribute as a keyword to retrieve general scenarios. The architect can then decide to instantiate those general scenarios into concrete scenarios. These general scenarios can also help the architect to identify the patterns that can be used to satisfy their requirements. Moreover, those general scenarios can also lead the architect to identify a reasoning model that should be used to analyse

architectural decisions. In this process, the architect can use the different search features provided by PAKME.

The architect may decide to find out if similar problems have been solved in other projects. They can browse through the existing projects for similar problems. Having found a similar project, the architect can retrieve the architecture decisions taken, design options considered, rationale for choosing a certain design option, tradeoffs made, and findings of architecture evaluation. Such information can help the architect to decide whether the architecture decision can be reused or not, and how much tailoring is required. Project-specific knowledge can also help designers, developers and maintainers to better understand the architectural decisions, their constraints and reasoning behind it. Availability of the rationale behind the design decisions helps architects to explain architectural choices and how they satisfy business goals [5]. Such knowledge is also valuable during implementation and maintenance.

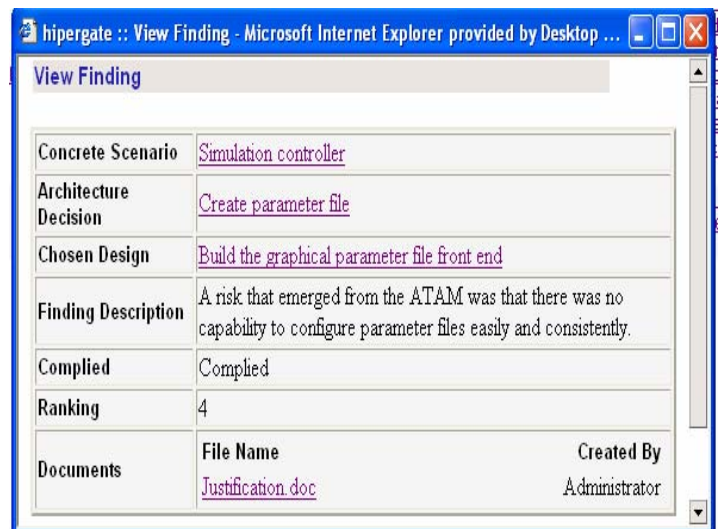
4. Support for design and analysis methods

To demonstrate PAKME's support for architecture design and analysis methods, this section discusses PAKME's use in the context of a generic model of architecture design reported in [13]. This model has three main activities: architectural analysis, architectural synthesis, and architectural evaluation. PAKME can be helpful in all three activities of this generic design model. For example, architectural analysis is aimed at eliciting architecturally significant requirements (ASRs), which are usually characterised by concrete scenarios. PAKME provides several hundred general scenarios (as shown in Figure 2), which can be concretised to specify quality attributes for a given system.

Architectural synthesis identifies candidate architectural solutions that address the ASRs elicited in the architectural analysis activity. PAKME provides a repository of generic design options, and architectural and design patterns that can be examined and assessed by an architect to compose an architectural decisions by tailoring existing design options, or selecting suitable styles, patterns, or tactics for building new design options.

Architectural evaluation attempts to ensure that the architectural decisions are the right ones. PAKME can support architecture evaluation in several ways. For example, if a method like ATAM [26] is used for evaluating an architecture, PAKME supports several activities (such as generating utility tree, identifying a reasoning framework, recording evaluation findings, and building a results tree to visualize risks and risk themes) of this method. During architecture evaluation, architecture knowledge captured by PAKME helps assess the suitability of certain patterns in the proposed architecture by matching the required concrete scenarios with the general scenarios extracted from the patterns used in the architecture as described in [31]. Moreover, PAKME helps evaluation team to capture findings from analysing architecture decisions and viewing the

justification for those finding. Figure 11 shows one finding from evaluating one architecture design decision. It shows the concrete scenario, proposed architecture decision, design option used, ranking of the decision relative to other proposed decisions, and any associated documents.



Concrete Scenario	Simulation controller	
Architecture Decision	Create parameter file	
Chosen Design	Build the graphical parameter file front end	
Finding Description	A risk that emerged from the ATAM was that there was no capability to configure parameter files easily and consistently.	
Complied	Complied	
Ranking	4	
Documents	File Name	Created By
	Justification.doc	Administrator

Figure 11: Evaluation findings captured in PAMKE

PAKME also provides templates for capturing rationale underpinning decisions as required by the three main activities of the generic model [13]. Moreover, provision of design, analysis, and realization knowledge is considered a critical input to the design process proposed in [13]. PAKME provides several types of design and analysis knowledge such as general scenarios, generic design decision, styles, patterns, tactics, and analytical frameworks. We are also confident that PAKME can support several of the ten techniques proposed in [14] for the SEI's methods for architecture analysis and design, however, space limitations do not allow us to provide any elaboration.

5. Current status and future work

Currently, we are trialing PAKME in an industrial architecture evaluation process, which requires organising large amounts of design knowledge. The introduction of PAKME is expected to help the industrial collaborator to systemise architecture evaluation process by managing the knowledge required for architecture evaluation. Logistical details and initial findings of this trial have been reported in [32].

We have been developing a Wiki-based component of PAKME to support collaborative decision making. We plan to implement more functions, such as templates for describing architecture provided by the Views and Beyond approach [28], and representation of architectural views in reporting functionality. PAKME does not supports diagrammatic modeling of design decisions rather its focus is on providing a handbook of architecture knowledge like the one being developed by

Booch [33] and suggested in [15]. However, we plan to explore the benefits and viability of integrating a repository of architecture knowledge like PAKME with commercial tools for modeling (like Enterprise Architect) as well as with research prototype like Archium [30]. A study for integration with Requirements management tool has also been planned.

Acknowledgement – *Lingzhi Xu helped us build PAKME. The authors led the tool development project while working with National ICT, Australia.*

7. References

- [1]. C. Potts and G. Bruns, Recording the Reasons for Design Decisions, *10th Int'l Conf. on Software Eng.*, 1988.
- [2]. J. Lee and K.-Y. Lai, What's in Design Rationale? *Human-Computer Interaction*, 1991. **6**(3-4): pp. 251-280.
- [3]. A.H. Dutoit and B. Paech, Rationale Management in Software Engineering, in *Handbook of Software Engineering and Knowledge Engineering*, S. Change, Editor. 2001, World Scientific Publishing, Singapore. pp. 1-29.
- [4]. T. Gruber and D. Russell, Design Knowledge and Design Rationale: A Framework for Representation, Capture, and Use, *Tech Report KSL 90-45*, Knowledge Laboratory, Stanford University, Stanford, United States, 1991.
- [5]. J. Tyree and A. Akerman, Architecture Decisions: Demystifying Architecture, *IEEE Software*, 2005. **22**(2): pp. 19-27.
- [6]. J. Bosch, Software Architecture: The Next Step, *European Workshop on Software Architecture*, 2004.
- [7]. F. Pena-Mora and S. Vadhavkar, Augmenting design patterns with design rationale, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 1997. **11**: pp. 93-108.
- [8]. M. Ali-Babar, I. Gorton, and B. Kitchenham, A Framework for Supporting Architecture Knowledge and Rationale Management, in *Rationale Management in Software Engineering*, A.H. Dutoit, et al., Editors. 2006, Springer. pp. 237-254.
- [9]. A. Jansen and J. Bosch, Software Architecture as a Set of Architectural Design Decisions, *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*, 2005.
- [10]. P. Kruchten, P. Lago, and H.V. Vliet, Building up and Reasoning about Architecture Knowledge, *Proceedings of the 2nd International Conference on Quality of Software Architectures*, 2006.
- [11]. A. Tang, M. Ali-Babar, I. Gorton, and J. Han, A Survey of Architecture Design Rationale, *Journal of Systems and Software*, 2006. **79**(12): pp. 1792-1804.
- [12]. M. Ali-Babar, I. Gorton, and R. Jeffery, Toward a Framework for Capturing and Using Architecture Design Knowledge, *Tech Report TR-0513*, University of New South Wales, Australia, 2005.
- [13]. C. Hofmeister, P. Kruchten, R.L. Nord, H. Obbink, A. Ran, and P. America, A General Model of Software Architecture Design Derived from Five Industrial Approaches, *the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA 05)*, Pittsburgh, PA, USA, 2005.
- [14]. R. Kazman, L. Bass, and M. Klein, The essential components of software architecture design and analysis, *Journal of Systems and Software*, 2006. **79**: pp. 1207-1216.
- [15]. G. Arango, E. Schoen, and R. Pettengill, A Process for Consolidating and Reusing Design Knowledge, *Proceedings of the 15th International Conference on Software Engineering*, 1993.
- [16]. Hipergate. An open source CRM and Groupware system. Last accessed on 16th March, 2006, Available from: <http://www.hipergate.com>.
- [17]. B.A. Kitchenham, R.T. Hughes, and S.G. Linkman, Modeling software measurement data, *Software Engineering, IEEE Transactions on*, 2001. **27**(9): pp. 788-804.
- [18]. M.T. Hansen, N. Nohria, and T. Tierney, What's Your Strategy For Managing Knowledge? *Harvard Business Review*, 1999. **March-April**: pp. 106-116.
- [19]. K.C. Desouza and J.R. Evaristo, Managing Knowledge in Distributed Projects, *Communication of the ACM*, 2004. **47**(4): pp. 87-91.
- [20]. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. 2 ed. 2003: Addison-Wesley.
- [21]. B. Skuce, Knowledge management in software design: a tool and a trial, *Software Engineering Journal*, Sept. 1995: pp. 183-193.
- [22]. L.G. Terveen, P.G. Selfridge, and M.D. Long, Living Design Memory: Framework, Implementation, Lessons Learned, *Human-Computer Interaction*, 1995. **10**(1): pp. 1-37.
- [23]. S. Henninger, Tool Support for Experience-Based Software Development Methodologies, *Advances in Computers*, 2003. **59**: pp. 29-82.
- [24]. D. Alur, J. Crupi, and D. Malks, *Core J2EE Patterns: Best Practices and Design Strategies*. 2nd ed. 2003: Sun Microsystems Press.
- [25]. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*. 1996: John Wiley & Sons.
- [26]. P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*. 2002: Addison-Wesley.
- [27]. L. Bass, M. Klein, and F. Bachmann, Quality Attribute Design Primitives, *Tech Report CMU/SEI-2000-TN-017*, SEI, Carnegie Mellon University, USA, 2000.
- [28]. P. Clements, et al., *Documenting Software Architectures: Views and Beyond*. 2002: Addison-Wesley.
- [29]. J.L. Kolodner, Improving Human Decision Making through Case-Based Decision Aiding, *AI Magazine*, 1991. **12**(2): pp. 52-68.
- [30]. A. Jansen, J.v.d. Ven, P. Avgeriou, and D. Hammer, Tool Support for Architectural Decisions, *Proceedings of the 6th working IEEE/IFIP Conference on Software Architecture, Mumbai, India*, 2007.
- [31]. M. Ali-Babar, I. Gorton, and R. Jeffery, Capturing and Using Software Architecture Knowledge for Architecture-Based Software Development, *Proceedings of the 5th International Conference on Quality Software*, 2005.
- [32]. M. Ali-Babar, A. Northway, I. Gorton, P. Heuer, and T. Nguyen., Introducing Tool Support for Knowledge Management in Software Architecture Evaluation Process, *Tech Report National ICT*, Australia, 2007.
- [33]. G. Booch. *Handbook of Software Architecture*. Last accessed on 16th January, 2007, Available from: <http://www.booch.com/architecture/blog.jsp>.