Introducing Tool Support for Knowledge Management in Software Architecture Evaluation Process

[1]Muhammad Ali Babar, [2]Andrew Northway, [3]Ian Gorton, [2]Paul Heuer, [2]Thong Nguyen
[1]Empirical Software Engineering, National ICT Australia Ltd.
malibaba@nicta.com.au
[2]Air Operations Division, Defence Science and Technology Organisation
{andrew.northway,paul.heuer,thong.nguyen}@dsto.defence.gov.au
[3]Pacific Northwest National Laboratory, USA
Ian.gorton@pnnl.gov

NICTA Technical Report # PA006116.
January 2007

Abstract

Management of software architecture knowledge is vital for improving an organisation's architectural capabilities. Despite the recognition of the importance of capturing and reusing software architecture knowledge, there is currently no suitable support mechanism. To address this issue, we have developed a conceptual framework for managing architecture design knowledge. A web-based knowledge management tool, Process-based Architecture Knowledge Management Environment (PAKME), has been developed to support that framework. This report discusses the main architectural components and features of PAKME. We also discuss different usages of the tool for capturing and using architecture design knowledge to support the software architecture process. This report also describes the objectives, logistics and initial findings of deploying and trialling PAKME in an Australian Defence acquisition environment for evaluating a military mission system's architecture.

Table of Content

1. Introduction

The Software Architecture (SA) process consists of several activities (such as design, documentation, and evaluation), which involve complex knowledge intensive tasks [1, 2]. The knowledge that is required to make suitable architectural choices and to rigorously assess those design decisions is broad, complex, and evolving. Such knowledge is often beyond the capabilities of any single architect. The software architecture community has developed several methods (such as a general model of software architecture design [3], Architecture Tradeoff Analysis Method (ATAM) [4], and architecture-based development [5]) to support a disciplined architecture process. Although, these approaches help manage complexity by using systematic approaches, they provide little support to provide or manage the knowledge required or generated during the software architecture process.

The requisite knowledge can be technical (such as patterns, tactics, and quality attribute analysis models) or contextual (such as design options considered, tradeoffs made, assumptions, and design reasoning) [6]. The former type of knowledge is required to identify, assess, and select suitable design options for design decisions. The latter is required to provide the answers about a particular design option or the process followed to select that design option [7, 8]. If not documented, knowledge concerning the domain analysis, patterns used, design options evaluated, and decisions made is lost, and hence is unavailable to support subsequent decisions in development lifecycle [9-11].

Recently, various researchers [12, 13] have proposed different ways to capture contextual knowledge underpinning design decisions. An essential requirement of all these approaches is to describe software architecture in terms of design decisions and the knowledge surrounding them. However, architecture design decisions and the contextual knowledge are seldom documented in a rigorous manner. Moreover, we have also found that there is little use/reuse of the architectural artefacts (such as scenarios, quality attributes and tactics) informally described in patterns' documentation [14]. This shortfall is simply because current formats for describing patterns are not suitable for the software architecture process – too much detail is counter-productive as expert designers usually follow a breadth-first and depth-later approach [1]. Nor do pattern documentation formats explicate the schemas of the relationships among scenarios, quality attributes, and patterns in a way that makes this knowledge readily reusable.

In order to provide an infrastructure for managing software architecture knowledge, we have developed a framework for managing architecture knowledge [6, 15]. This framework consists of techniques for capturing design decisions and contextual information, an approach to distill and document architectural information from patterns, and a data model to characterise architectural constructs, their attributes and relationships [6]. The central objective of this framework is to provide a theoretical underpinning and conceptual guidance to design and implement a repository-based tool support for managing architecture knowledge [15]. We have extended this framework to incorporate Case-Based Reasoning (CBR) to contextualise the captured knowledge. The novelty of this framework resides in its ability to incorporate all the components into an integrated approach, which has been implemented in a web-based architecture knowledge management tool called PAKME (Process-based Architecture Knowledge Management Environment). This report describes various aspects of PAKME and explains how PAKME has been applied in the context of

evaluating architectures for mission systems in the Defence environment. The implementation of PAKME is intended to provide a practical solution to knowledge management issues that characterise the architecture process and are discussed in the following sections.

2. Knowledge Management Problems in Architecture Process

One of the problems in software architecting process is the lack of access to knowledge underpinning the design decisions and process [6, 10]. This type of knowledge involves things like the impact of certain middleware choices on communication mechanisms, why an API is used instead of a wrapper, and who to contact to discuss the performance issues. Much of this knowledge is episodic and is usually not documented [2]. The absence of a disciplined approach to managing architecture knowledge has many downstream consequences which 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 artefacts and process.

All of these factors cause a loss of substantial knowledge generated during the architecture process, thus depriving organisations of a valuable resource. Further, loss of key personnel may mean loss of knowledge [2, 16, 17].

The architecture research community has developed several methods (such as ATAM [4], PASA [18]) to support a disciplined approach to architectural practices. Some of these methods emphasise the need for knowledge management. However, there is no approach that explicitly states what type of knowledge needs to be managed and how, when, where, or by whom to support architecture activities. Also, none of the current approaches provides any conceptual framework to design, develop and maintain a repository of architecture knowledge. To address these issues, we have developed a framework for managing software architecture knowledge. This framework incorporates concepts from knowledge management [19, 20], experience factories [21, 22], and pattern-mining [14, 23] paradigms to provide an integrated support environment. The framework requires a knowledge repository, which is logically divided into two types of architecture knowledge: generic architecture knowledge (such as general scenarios, patterns, quality attributes, design options and others), and project specific architecture knowledge (such as concrete scenarios, contextualised patterns, quality factors, architectural decisions and others). The generic knowledge is accumulated by capturing architecture knowledge using the techniques included in our framework for capturing and using architecture design knowledge [15].

3. 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 [24]. This platform provides various collaborative features including contact management, project management, online collaboration tools and others, which can be exploited to build a groupware support

mechanism, which incorporate architecture knowledge management features, for geographically distributed stakeholders involved in the software architecture process.
An appropriate conceptual data model is a prerequisite for developing an integrated support environment to assist in the improvement of a certain software development process [25] such as software architecting.

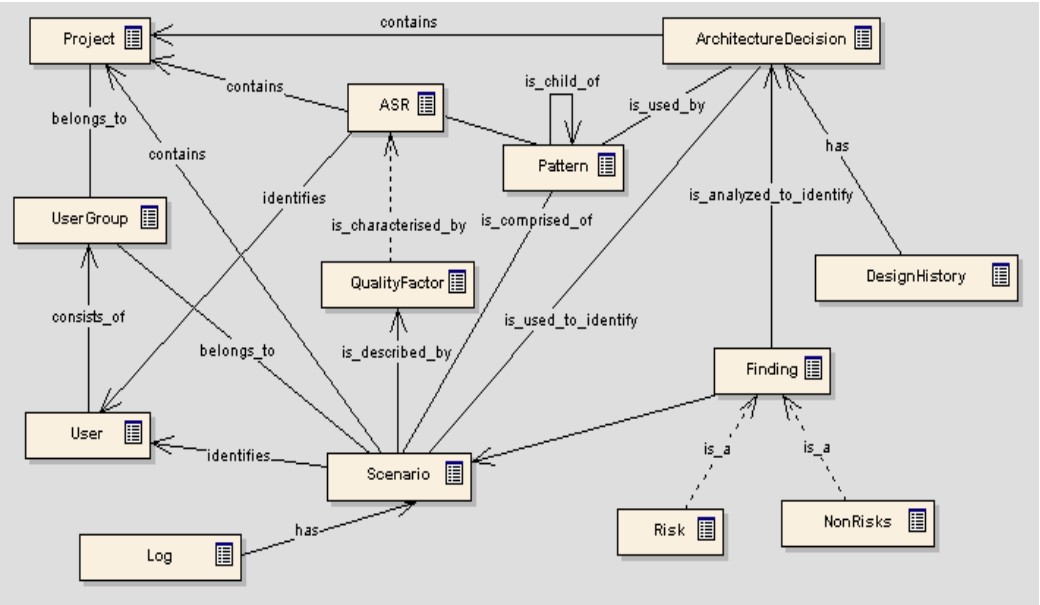


Figure 1: A partial data model characterising architectural artefacts captured in the software architecture knowledge base

Like the modelers of measurement data [26], 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 underpin the architecture design knowledge. Additionally, the data model needs to be customisable in order to meet the needs of different domains and organisations for characterising architecture knowledge [6]. Figure 1 presents a partial conceptual data model that identifies the main architectural constructs and their relationships. This data model has been constructed during domain modeling aimed at characterizing knowledge used or generated during software architecture process. The process of domain modeling and an extended version of this data can be found in [6].
The knowledge repository of PAKME is logically divided into organisational knowledge (generic), and project-specific knowledge (concrete). The generic architecture knowledge is accumulated by using the knowledge capture techniques described [27, 28]. Project-based architecture knowledge consists of the artifacts either instantiated from the generic knowledge or newly created during different activities of the software architecture process. Access to a repository of generic architecture knowledge enables designers to use the accumulated “wisdom” from different projects when devising or analysing architectural decisions for projects in the same or similar domains. For example, instantiating abstract scenarios into concrete ones, contextualizing design decisions and others. The project specific part of the data model also has other entities to capture and consolidate architecture

knowledge and rationale that is specific to a particular project. For example, design history, findings of architecture analysis, architectural views of interest to each type of stakeholders and others. A project specific architecture knowledge repository is populated with architecture knowledge drawn from the organisational repository, standard work products of the design process, logs of the deliberations and histories of documentation to build organization's architecture design memory [29].

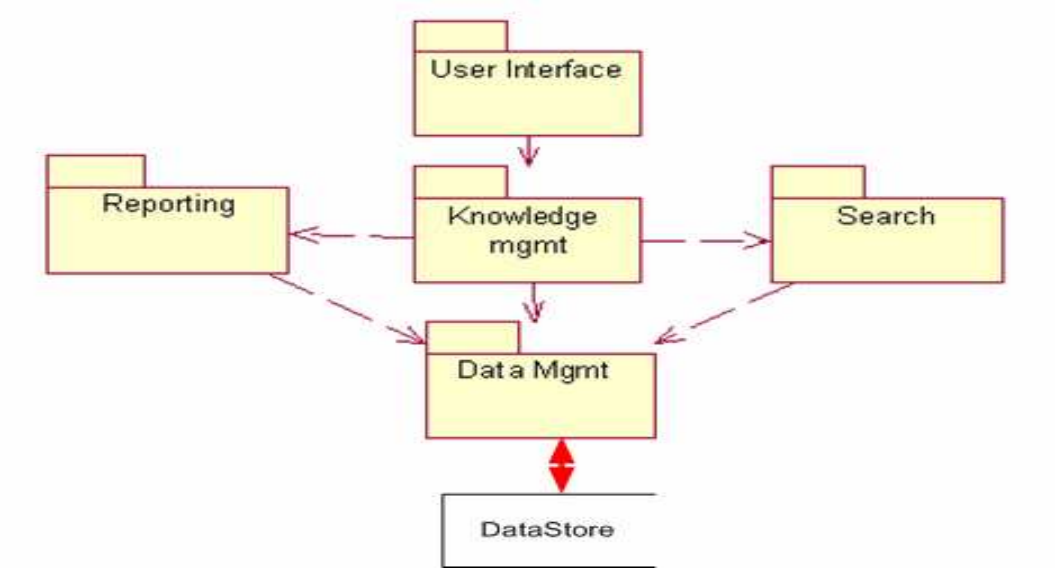


Figure 2: Component view of PAKME' architecture

We have also made certain modifications to the presentation and business logic tiers of Hipergate in order to add the features required to capture, retrieve and manage architecture design knowledge. Currently PAKME consists of four components as shown in Figure 2.

- User Interface – The only way to interact with PAKME is through a user interface that is integrated into Hipergate. It has been implemented using Java Server Pages (JSP) and HTML.
- Knowledge management – This component provides common services to store, retrieve, and update artefacts that make up architecture knowledge. This component also provides services to instantiate generic architectural artefacts into project-specific artefacts. For example, creating concrete scenarios to characterise quality attributes for a project based on general scenarios stored in the knowledge-base.
- 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. The keyword-based search facility explores the repository for a desired artefact utilising key words that are attached as meta-data to each artefact. The advanced search function is based on a combination of logical operators. The navigational search is provided by presenting the retrieved artefacts as hyperlinks, which can be traversed for further details.
- 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. The reporting component also supports architecture evaluation by helping stakeholders develop utility trees to specify quality attributes and presenting the findings of architecture analysis as a result tree.

- Data 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. The repository has been designed based on a data model described in [6] and partially presented in Figure 1.

4. Managing Architecture Knowledge with PAKME

Most of the approaches to managing knowledge can broadly be categorized into codification and personalization [30]. Codification concentrates on identifying, eliciting and storing knowledge as information in repositories, which are expected to support high-quality, reliable, and speedy reuse of knowledge. Personalization resorts to fostering interaction among knowledge workers for explicating and sharing knowledge. Though, this paper focuses on those 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 to managing knowledge [31]. Here we briefly discuss the four main services of PAKME for managing architecture knowledge to support software architecture process:

- 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.
- 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.
- 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.
- Knowledge presentation service helps presents knowledge to in a structured manner at a suitable abstraction level by using templates (such as provided in [32]) and representation mechanisms like utility and results trees described in [33].

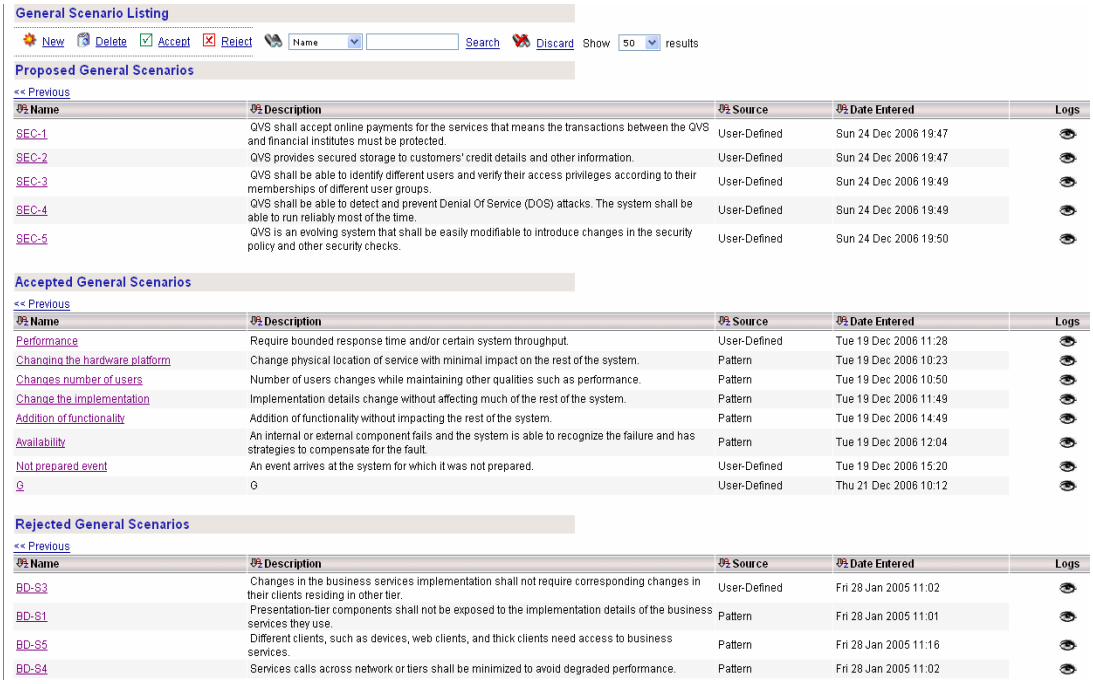
These services not only satisfy the requirements identified by us to provide knowledge management support for methods like [34, 35], but also support many of the use cases proposed in [13] such as add a decision, retrieve a design decision, get a rationale, clone architecture knowledge, attach relevant documents to artifacts, and study the chronology of design decisions.

4.1. Capturing and presenting knowledge

There are usually two main strategies to elicit and codify knowledge:

1. Appoint a knowledge engineer to elicit and codify knowledge from individuals or teams [2, 36];
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 [37]; each of this strategy has its strengths and weaknesses. To take the advantage of 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 [38] patterns and architecture patterns [39], and case studies described in [4, 33] or design primitives [40].



General Scenario Listing

New Delete Accept Reject Name Search Discard Show 50 results

Proposed General Scenarios

<< Previous

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	
SEC-2	QVS provides secured storage to customers' credit details and other information.	User-Defined	Sun 24 Dec 2006 19:47	
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	
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	
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	

Accepted General Scenarios

<< Previous

Name	Description	Source	Date Entered	Logs
Performance	Require bounded response time and/or certain system throughput.	User-Defined	Tue 19 Dec 2006 11:28	
Changing the hardware platform	Change physical location of service with minimal impact on the rest of the system.	Pattern	Tue 19 Dec 2006 10:23	
Changes number of users	Number of users changes while maintaining other qualities such as performance.	Pattern	Tue 19 Dec 2006 10:50	
Change the implementation	Implementation details change without affecting much of the rest of the system.	Pattern	Tue 19 Dec 2006 11:49	
Addition of functionality	Addition of functionality without impacting the rest of the system.	Pattern	Tue 19 Dec 2006 14:49	
Availability	An internal or external component fails and the system is able to recognize the failure and has strategies to compensate for the fault.	Pattern	Tue 19 Dec 2006 12:04	
Not prepared event	An event arrives at the system for which it was not prepared.	User-Defined	Tue 19 Dec 2006 15:20	
G	G	User-Defined	Thu 21 Dec 2006 10:12	

Rejected General Scenarios

<< Previous

Name	Description	Source	Date Entered	Logs
BD-S3	Changes in the business services implementation shall not require corresponding changes in their clients residing in other tier.	User-Defined	Fri 28 Jan 2005 11:02	
BD-S1	Presentation-tier components shall not be exposed to the implementation details of the business services they use.	Pattern	Fri 28 Jan 2005 11:01	
BD-S5	Different clients, such as devices, web clients, and thick clients need access to business services.	Pattern	Fri 28 Jan 2005 11:16	
BD-S4	Services calls across network or tiers shall be minimized to avoid degraded performance.	Pattern	Fri 28 Jan 2005 11:02	

Figure 3: General scenarios captured by PAKME's repository

As we mentioned, PAKME provides several kinds of 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 [35]. Figure 4 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 3 shows some of them).

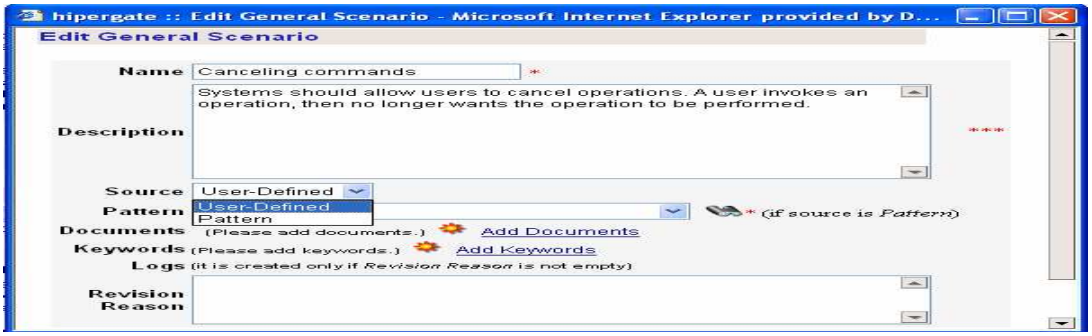


Figure 4: The interface to capture a general scenario

Figure 5 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 (i.e., architectural pattern containing design patterns) and each pattern may have several tactics attached to it. To support the 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 during architecture design. 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 composed of one or more tactics. For example, publish-scribe design option applies publish-on-demand design pattern.

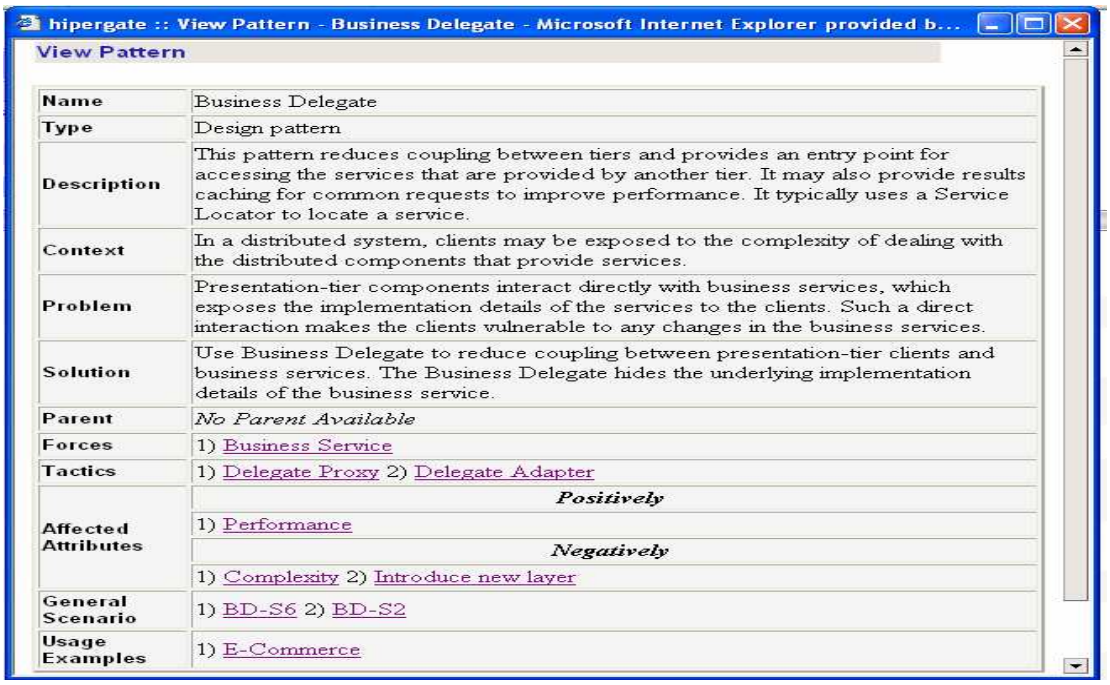


Figure 5: Template to capture and present patterns

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 is captured in a separate template, which is designed based on practitioners' opinion about rationale reported in [41] and templates proposed in [9, 42]. Figure 6 shows a partial description of a design option. By capturing design options as cases, PAKME enables architects to follow a case-based decision approach and supports human-intensive case-based reasoning [43].

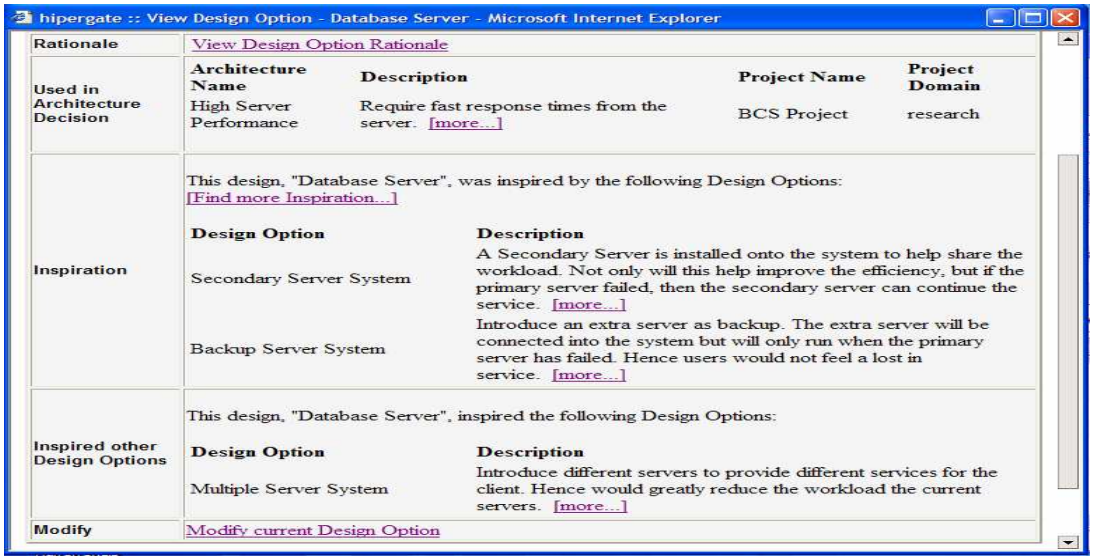


Figure 6: A partial view of a design option case

Recently, there has been an increased emphasis on describing software architecture as a set of design decisions [10, 44]. Kruchten et al. have proposed a taxonomy of architectural decisions, their properties, and relationships among them [13]. Figures 7 shows that PAKME can capture many of the attributes and relationships of architecture design decision as described in [13] and template proposed in [9]. In PAKME, architecture design decision can be described at different levels of granularity as an architecture design decision is a selected design option, which can be composed of architectural pattern, design pattern or design tactic. Like a design option, each architecture design decision also captures rationale using a template. This rationale describes the reasons underpinning the architecture design 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.

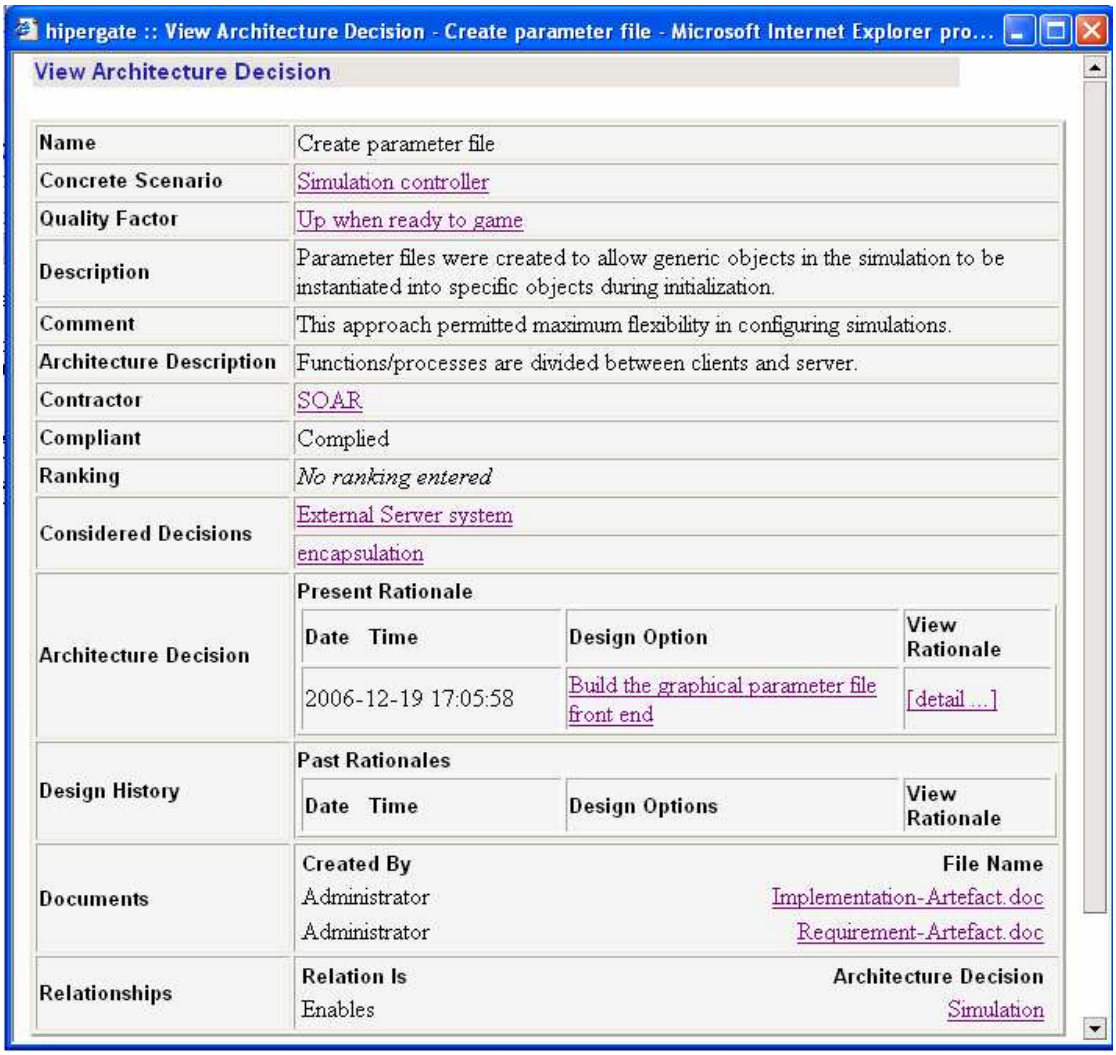


Figure 7: An architecture decision captured in PAKME

Moreover, traceability is also provided as each architectural design decision describes the design options considered but rejected, concrete scenarios to be satisfied, and model of architectural decision attached as design artifacts (Shown in Figure 7). Additionally, revisions to architecture design decisions and reasons are logged for later review. Architecture design decisions are time stamped and annotated with the decision maker's details, which can be used to seek further explanation for an architectural design decision. Hence, we believe that PAKME supports the description of an architecture design decision in ways suggested in [9, 44] and the attributes and relationships proposed in [13]. Figure 8 shows that a user can establish several types of relationships among architecture design decisions.

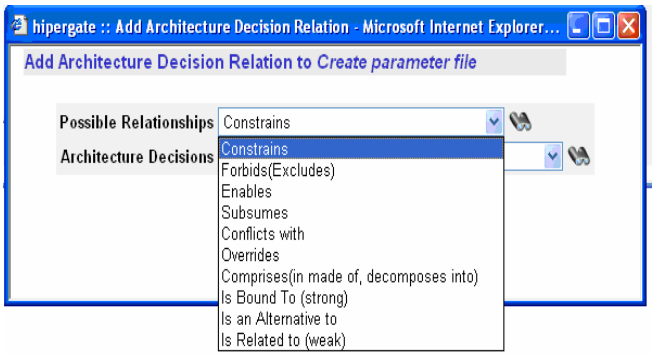


Figure 8: Types of relationships that can be established

4.2. 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 9 shows that there is a four steps process of reusing design options, which are captured as cases.

This process starts when a user has a new requirement that needs architectural support. Such 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.

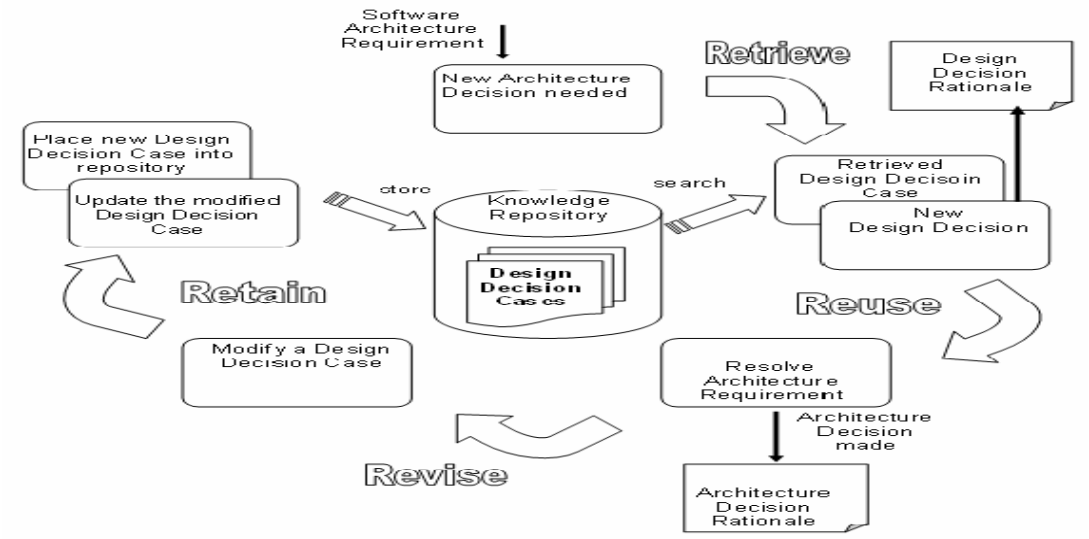


Figure 9: Process model of reusing design options

If the architect decides to search through the knowledge repository for cases of design options, he/she can perform a search to retrieve a list of design options. Figure 10 shows that a user can build a very 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 modifies it according to the current context. Figure 11 shows that a retrieved design option can be used/reused 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 attachment.

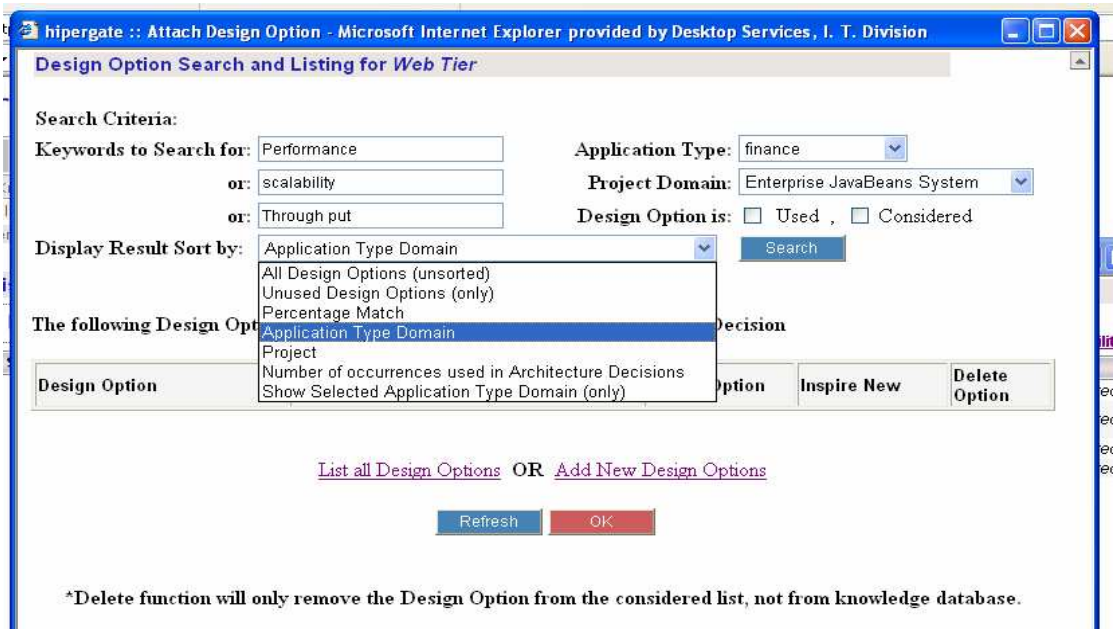


Figure 10: PAKME's interface for searching design option cases that can be used in architecture design

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 that should satisfy certain non-functional requirements. The architect is likely to make architectural decisions using a common process - understanding the problem, identifying potential alternatives, and assessing their viability. There are a few ways PAKME can support this process. The architect can search the repository for architectural artefacts that can be reused. For example, he/she can use a particular quality attribute (e.g. performance) as a keyword to retrieve general scenarios that characterise performance. The architect decides to instantiate those general scenarios into concrete performance scenarios. These general scenarios can also help the architect to identify the patterns that can be used to satisfy the performance 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 different search features provided by PAKME.

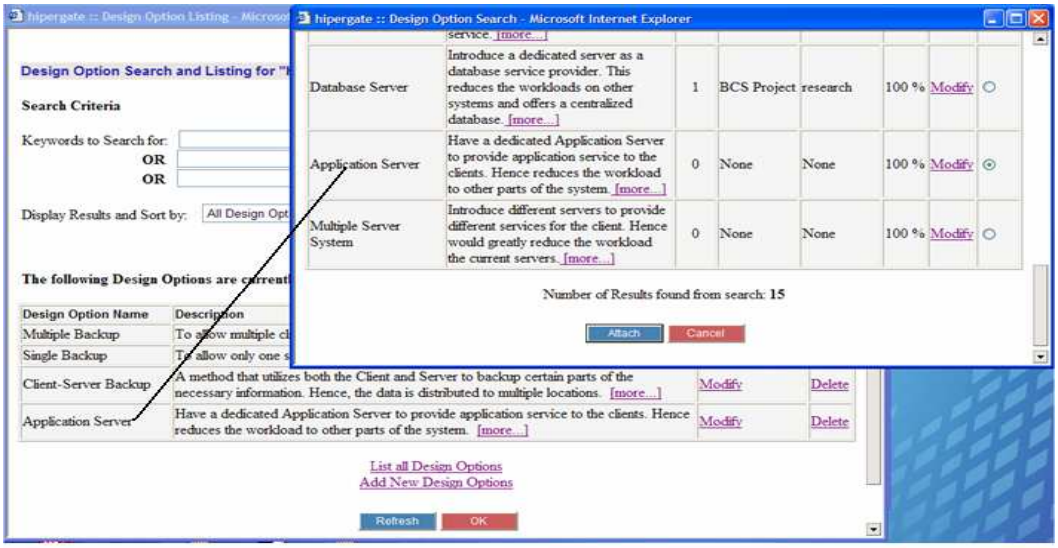


Figure 11: Attaching a retrieved design option to an architecture design

The architect may decide to find out if similar problems have been solved in other projects. He/she can browse through the existing projects for similar problems. Having found a similar project, the architect can retrieve the architecture design decisions taken, design options considered, rationale for choosing a particular 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. Additionally, 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 reasoning behind the design decisions helps architects to explain architectural choices and how they satisfy business goals [9]. Such knowledge is also valuable during implementation and maintenance stages.

4.3. Support for design and analysis methods

In order to understand how PAKME can support a particular method of architecture design and/or analysis. Let us consider PAKME's use in the context of a generic model of architecture design recently proposed by Hofmeister et al. [34]. This model has three main activities: architectural analysis, architectural synthesis, and architectural evaluation. We believe that 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 hundreds of general scenarios (as shown in Figure 3), which can be concretised to specify quality attributes for a given system.

Architectural synthesis intends to identify candidate architectural solutions that address 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 used are the right ones. PAKME can support architecture evaluation in several ways. For example, if a method like ATAM [4] is used for evaluating software architecture, PAKME provides different features to supports several activities (such as generating utility tree, identifying suitable reasoning framework, recording evaluation findings, and building results tree to visualize risks and risk themes) of this method. During software 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 [15]. Moreover, PAKEM helps evaluation team to capture findings from analysing architecture decisions and justification for those finding. Figure 12 shows one finding from evaluating one of the 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. Apart from temple-based presentation of findings, PAKME also generates PDF-based reports of findings.

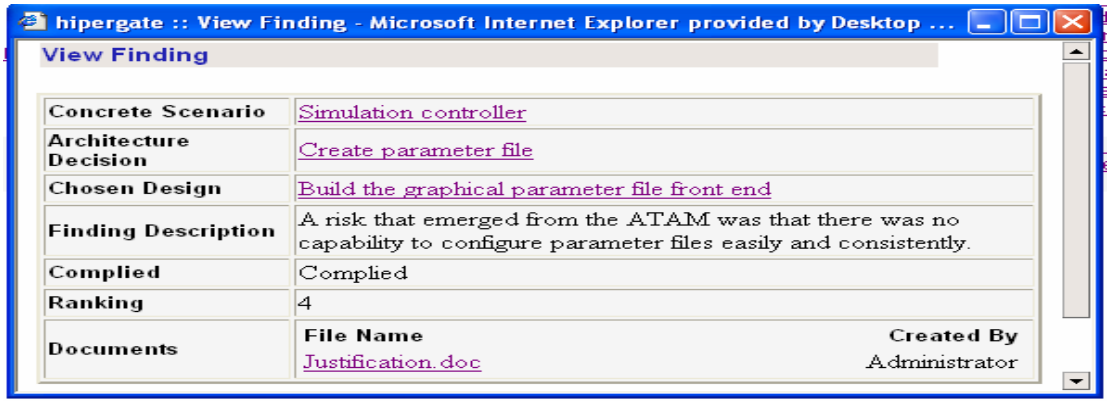


Figure 12: Evaluation findings captured in PAMKE

PAKME also provides template for capturing rationale underpinning decisions as required by the three main activities of the generic model [34]. Moreover, provision of design, analysis, and realization knowledge is considered a critical input to the design process proposed in [34]. PAKME provides several types of design and analysis knowledge such as general scenarios, generic design decision, styles, patterns, tactics, and analytical frameworks.

Apart from supporting well-known methods and approaches incorporated into the generic model of architecture design as discussed in the previous section, PAKME's provides architectural knowledge management support to several of the ten techniques proposed in [35] for the SEI's methods for architecture analysis and design. For example, PAKME provides several templates to capture information during architecture analysis. Provision of suitable templates is important for making a method consistent across evaluators. Templates also help in consistently gathering and documenting information that is useful for the stakeholders [35]. The use of quality attribute scenarios is one of the core techniques for SEI's methods to characterize stakeholders' concerns. "General scenarios" are used to aid in

the elicitation of “concrete scenarios” using a six part framework as described in following paragraph.

PAKME provides a repository of domain-specific general scenarios (Shown in Figure 3 and 15) that are used to steer the process of developing concrete scenarios; PAKME also provides a template to capture the concrete scenarios. This template is based on six parts framework proposed in [33] but it only utilizes four parts (i.e., stimulus, source of stimulus, context, and response). Moreover, PAKME also helps stakeholders to structure and prioritise concrete scenarios using techniques like utility tree as shown in Figure 17. Explicit elicitation of architecture documentation and rationale in standardized views is another important technique to support architecture analysis and design [35]. PAKME supports the elicitation and capture of rationale for design decision by providing a template build upon the elements of design rationale reported in [9, 41]. Additionally, templates have been implemented to describe architecture decision at various levels of abstractions and each design decision may be composed of architectural or design patterns and tactics, which are represented using templates proposed in [4, 6]. We have already discussed how PAKME can support different activities of architecture evaluation using SEI’s architecture evaluation ATAM in the context of the generic model of architecture design.

5. Trialing PAKME

To demonstrate the use of PAKME for capturing and managing architecture design knowledge and rationale for improving the software architecture process, this section reports on an industrial trial of PAKME in the military mission system domain. This trial is a part of a research and development collaborative project being carried out by the National ICT Australia and the Defence Science and Technology Organisation (DSTO), Australia. This collaborative project is aimed at exploiting the architecture evaluation technologies developed by NICTA for improving DSTO’s capabilities in evaluating architectural risk during system acquisition. The case study reported here was undertaken to tailor and deploy PAKME in one of the divisions of DSTO for codifying and managing process and domain knowledge of evaluating software architecture.

5.1. Organisational context

DSTO is a research and development organisation, which provides scientific and technical advice on the acquisition of materiel to the Australian Defence Organisation (ADO). One of the key responsibilities of DSTO is to evaluate Request for Proposal (RFP) responses from tenderers to identify technical and project risks of each proposal. The Airborne Mission Systems (AMS) division of DSTO is responsible for evaluating software architectures for aircrafts acquisition projects. AMS is required to understand and organise large amount of architecture design knowledge for a mission system’s architecture to support the evaluation process. Currently there is a lack of a rigorous process for evaluating architectures. The architectural evaluation mainly relies on the domain knowledge of local experts. As the modern mission systems are increasingly becoming more reliant on software, evaluating proposed architectural solutions has become much more important as the software intensive projects are historically considered the most risk prone in the Defence domain [45]. Hence, there has been growing recognition of the importance of systemising architecture evaluation

architecture evaluation processes within Defence.

Recently, AMS's technical leadership has become increasingly interested in building its capabilities in systematically evaluating system and software architectures and managing architecture knowledge for aircraft mission systems. Hence, AMS has decided to improve its architectural evaluation practices by codifying and reusing an architecture evaluation process, architecture design knowledge, and contextual knowledge. This is expected to be achieved through the use of a tool like PAKME that can help AMS to capture and manage architecture knowledge.

5.2. Trial's objectives

This trial was a part of an ongoing collaboration between NICTA and DSTO aimed at improving AMS's architecture evaluation capabilities by capturing and managing organisational knowledge concerning system architecture evaluation and processes. It is expected that the use of an architecture knowledge management tool will systemise the process and help organise the architecture design knowledge and contextual information required or generated during a software architecture evaluation process.

This objective is expected to be achieved by embedding PAKME in the software architecture evaluation process. A simplified illustration of how PAKME has been embedded in the AMS's architecture evaluation framework is shown in Figure 13. Once integrated in the evaluation process as shown in Figure 13, PAKME supports several architecture evaluation tasks. For example, it helps build quality models using scenarios (abstracts and concrete), reason about the suitability of various design options proposed by contractors, capture the rationale for ranking, approving, or rejecting various design proposals, and centralise architecture design knowledge.

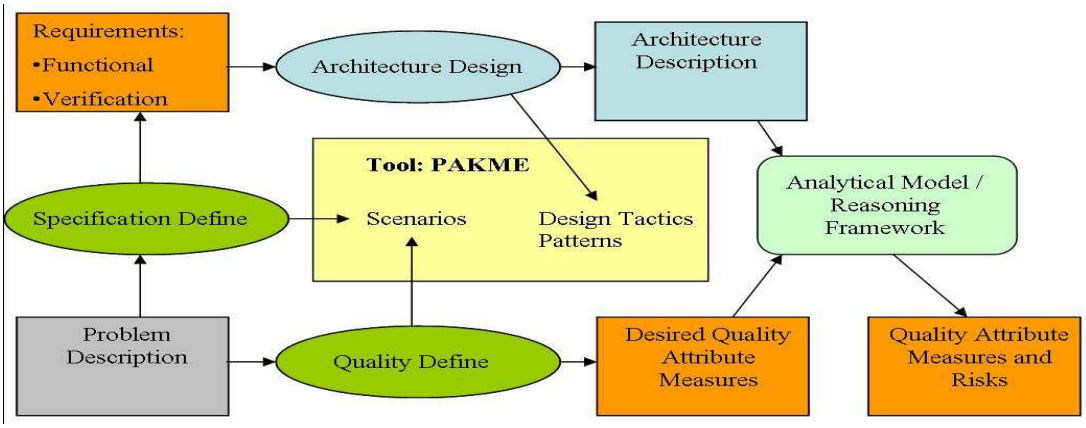


Figure 13: AMS's software architecture evaluation process supported PAKME

Although knowledge management initiative requires considerable time and resources, it is anticipated there will be considerable time and cost savings in the long-term [46]. DSTO and NICTA have also identified several benefits from managing architecture knowledge during architecture evaluation for Defence acquisition. Some of these benefits are:

- Capture rationale for architecture decisions

- Help build architectural capabilities
- Improve architectural reusability
- Provide an audit trail for TRA findings
- Reduce demands on subject matter experts
- Encourage best architectural practices
- Improve efficiency of architectural processes
- Accelerate the training process of new employees within the organisation

5.3. Tailoring PAKME

PAKME provides a generic solution to address the architecture knowledge management issues during the software architecture process. It is designed to help users access or capture architecture knowledge required or generated during software architecture design, documentation, or evaluation activities. Hence, it needs to be customised depending on the organisational requirements and role in the software architecture process. For example, AMS does not design or document architectures. Rather, it evaluates architectures proposed by contractors. Thus, it needed features of the tool that support software architecture evaluation tasks.

Therefore, there was a need to customise PAKME for supporting the AMS's architecture evaluation process. Initially analysis of the AMS's process also revealed the need for extra features and certain modifications to fulfill the requirements of the Defence environment. In order to identify the requirements for tailoring PAKME, a workshop was held. During this workshop, staff organisations collaboratively identified the initial set of requirements, which needs to be satisfied by PAKME to be applicable to AMS's architecture evaluaiton process. In addition to the requirements gathered during the worksop, AMS also generated additional set of requirements. All the requirements were categorised as high and low priority based their importance to the AMS's process needs. The high priority requirements were implemented in the current version of PAKME. Some of the high prirotiy requirements implemented for tailoring PAKME are:

- Classification of project data according to the Defence classification scheme
- Mechanism for recording compliance of architecture decisions with respect to requirements
- File-based report of findings
- Store and evaluate different tenderer's proposals for the same set of scenarios within the one project

Some of the lower priority requirements, which are being implemented include:

- Different levels of access to project data based on the Defence security scheme
- Ability to import/export data from the tool based on a classification code
- Risk management scheme for ranking design decisions
- Integration with requirements management and architecture modelling tools.

Both organisations have been equally participating in tailoring and enhancing PAKME for AMS's needs. NICTA's researchers and software engineers have been refining and implementing requirements. Whilst, AMS's staff members are testing the need features and reporting the bugs and errors back to the NICTA's team. In tis process, AMS has also

identified sevearl new requirements, which are expected to be included in the next phase of enhancing PAKME.

5.4. Project Description

In order to assess the use of PAKME for supporting the AMS's architecture evaluation process, a study was carried out within the AMS environment. This study involved using the PAKME for capturing and managing architecture knowledge to support architecture evaluation of an aircraft system. This study has been conducted as a post-mortem analysis of the architecture evaluation conducted without using PAKME. This study was aimed at investigating how the introduction of PAKME could help capture and manage architectural knowledge and whether or not the evaluation process is improved by using PAKME. Both organisations realised the need and importance of designing and conducting such a case study before deploying PAKME in the AMS's future evaluation projects.

A number of quality factors were chosen as measures for the mission system architecture evaluation process. The evaluation process involved DSTO analysts comparing alternative design decisions from multiple hypothetical tenders, to simulate the type of evaluation completed during the real evaluation of an aircraft acquisition project. The evaluation was performed by measuring each scenario against the quality attributes as well as assigning a measure of risk to the design solution. Some of the key architectural requirements of the system under evaluation included:

- An open system architecture
- Object-oriented software design
- POSIX-compliant layering
- Open standards-compliant
- Secure, open, and scaleable interfaces
- Hardware and software portability
- Support hardware and software failure detection, isolation, and recording
- Localisation and confinement of the effects of design changes and failures
- Provisions for adding more processing capability
- Data assurance and protection

5.5. Use of PAKME's knowledge base

PAKME's generic knoweldge base repository has been populated by generic architecturally signficant artefacts such as general scenarios, quality attributes, design options, design patterns, tactics, and analysis models. Most of the generic architectural knowledge comes with PAKME is more suitable to enterprise systems as such knoweldge has been captured based on the NICTA's experience in that domain. For example, Figure 14 shows a general scenario, which has been extracted from the data access object pattern using the pattern-mining process.

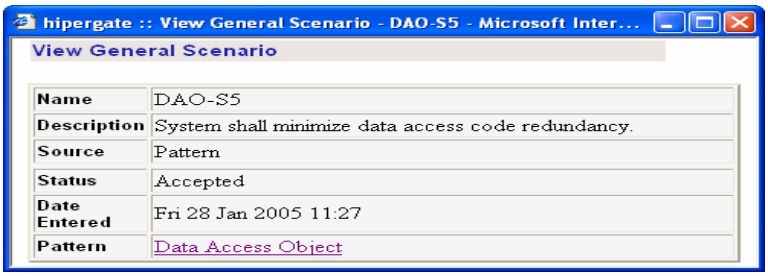


Figure 14: a general scenario distilled from a pattern

To populate the PAKME's knowledge base with the AMS's domain knowledge, AMS and NICTA held a workshop in April 2006 involving NICTA's researchers and AMS's domain experts. During this workshop, participants constructed a preliminary domain specific quality model for software architecture evaluation of Airborne Mission Systems (AMS). This quality model is based on ISO 9126 [47], SEI defined attributes, and AMS domain experience. The quality model involves identifying key quality attributes to enable evaluators to assess the potential risks of architectural designs against the requirements.

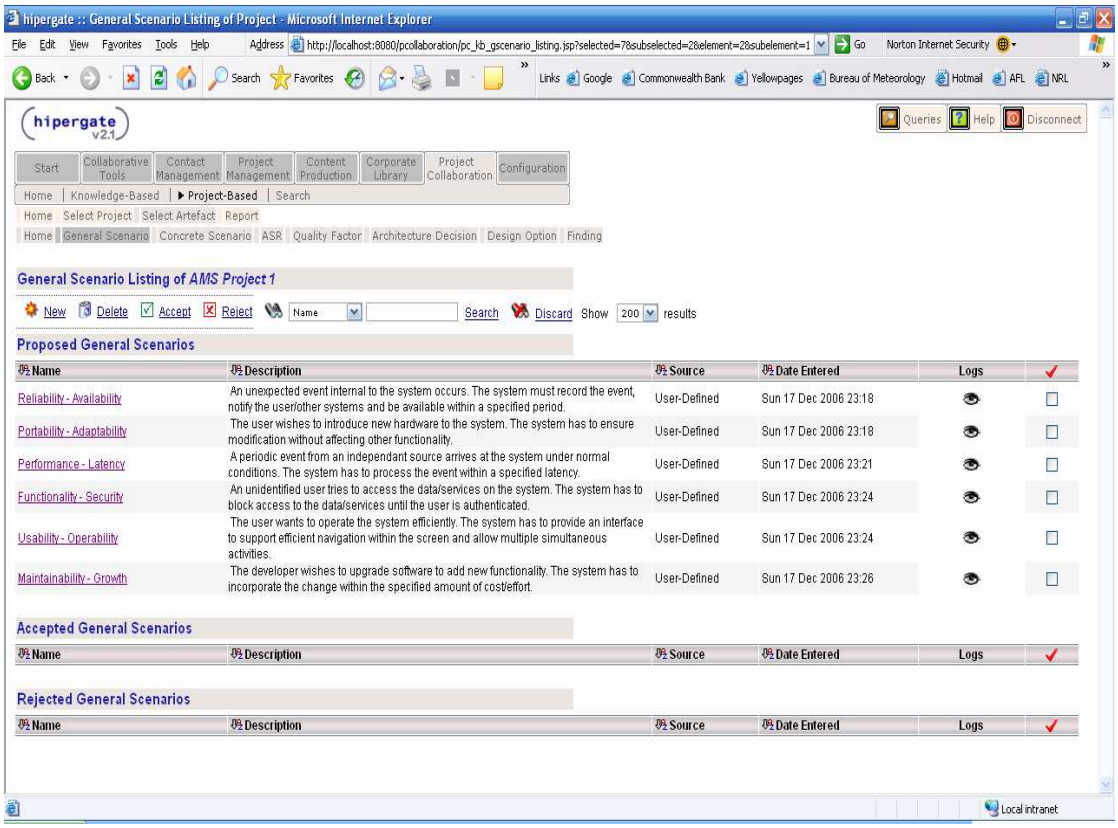


Figure 15: General scenarios captured by PAKME's repository

The qulaity model consists of into six quality attributes:

1. Performance
2. Reliability
3. Usability
4. Maintainability
5. Functionality
6. Portability

The quality model has been created in PAKME's repository for use by any AMS staff. PAKME's repository also has been populated with general scenarios for characterising each of the quality attribute included in the quality model. These scenarios have been developed and structured using a scenario development template provided by PAKME. This template has been designed based on the scenario development framework proposed by Bass et al in [33]. The general scenarios stored in PAKME are used to generate concrete scenarios for different evaluation projects of AMS. Figure 15 shows the general scenarios captured in PAKME to characterise a quality model for AMs.

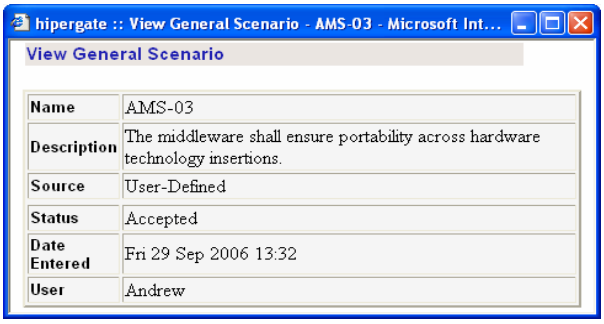


Figure 16: A user-defined general scenario

Figure 16 shows a general scenario for a generic mission system architecture used in this case study. PAKME's repository has also been populated with Defence specific general design options. These design options have been captured from the architecture solutions proposed for the system reviewed for this case study, AMS's domain experts, and case studies on avionics systems reported in sources like [33, 48]. Each design option has been captured as a design decision case as shown in Figure 4. These generic design options are used as input to design decision making or evaluation processes. The data captured in PAKME for this study have been sanitised of the sensitive and classified information about the aircraft system.

5.6. Use of PAKME's project base

PAKME's project base knowledge repository is used for capturing and managing project specific architecture knowledge such as quality factors, concrete scenarios, architecture decisions, rationale, and findings of evaluating architecture decisions. For this study, AMS's team create a new project in PAKME and populated its project-base with the project specific quality model to specify quality factors with concrete scenarios based on the general scenarios of the AMS's general quality model. Figure 17 shows a utility tree of the concrete

concrete scenarios that characterise quality factors growth, security and adaptability for this project.

Each architecture decision proposed by different contractors for satisfying required scenarios of the project was identified and entered into PAKME. Each architecture decision has been linked to the concrete scenario that is expected to be satisfied by that architecture decision. An example of a design decision affecting architectural quality is the use of a layered architecture including an isolation layer to reduce the impact of change, and thus improving flexibility, technology refreshment and growth capability. This architecture design decision has been stored in PAKME along the rationale. Each architecture decision of this project has also been captured as a design option in the generic knowledge base of PAKME. AMS's team has also captured several design options based on their domain knowledge. During architecture evaluation, each architecture design decision has been assessed with respective to the design options, which are expected to satisfy the same concrete scenario.

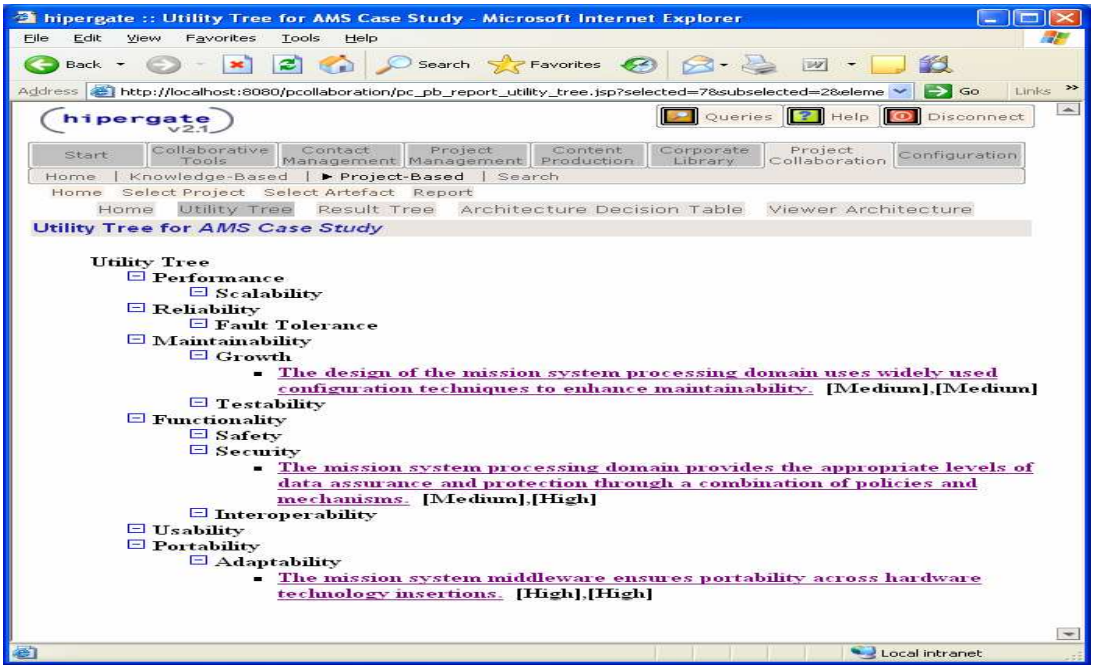


Figure 17: A utility tree for the case study

Having populated PAKME with the project specific architecture knowledge, AMS's team evaluated the architecture design decisions proposed for an aircraft system by several contractors. For this evaluation, AMS's team used PAKME for accessing the architecture knowledge required for the evaluation and capturing the findings and rationale for evaluation decisions. AMS's team used their existing process of evaluating architecture with one exception of introducing PAKME in the process as shown in Figure 13.

The architecture evaluation process involved determining whether or not the concrete scenario is satisfied by the proposed architecture decision. If there were more than one proposed architecture decision for a scenario, architecture decisions were assigned a ranking

based on the evaluator's opinion about each architecture decision's capability of achieving a certain level of required quality factor. Evaluators captured their findings in PAKME's repository. Each finding describes whether or not a certain architecture decision complied with the relevant requirement, its ranking, and rationale /justification underpinning the finding. Based on the evaluation findings, architecture decisions were categorised as risk and non-risks. Risks were further categorised under various risk themes.

Figure 18 shows a report of the findings of the evaluation carried out using PAKME. This report shows each concrete scenario and its associated architecture decision and findings. Apart from the browser based reporting, PAKME also generates PDF reports for evaluation teams and management.

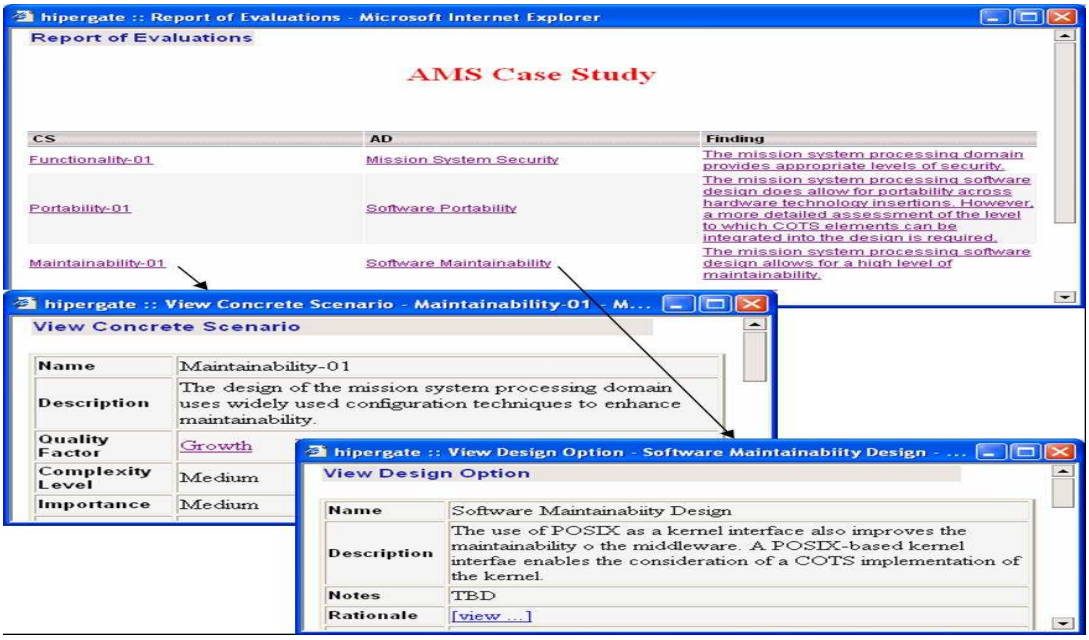


Figure 18: Reports of the evaluation case study

5.7. Challenges and observations

Customising PAKME for AMS proved to be a challenging task. NICTA's team did not have security clearance to access the architectures being evaluated by AMS. Nor did they have domain expertise. However, they had to gain certain level of domain understanding in order to help AMS generate domain-specific scenarios, identify and capture design options and patterns from architectural descriptions of the systems being evaluated, and determine requirements for customising PAKME. Quality attribute characterisation workshop held in April 2006 helped NICTA team to understand AMS's domain and identify initial set of requirements. During these workshops AMS's staff learned how to generate and structure general and concrete scenarios using templates provided by PAKME.

PAKME was not designed to classify data for security reasons. Nor did it handle the requirement of different levels of access to project data. For the first requirement, we had to make significant changes in PAKME's data structure. The other was handled by exploiting

Hipergate's domain and work area concepts for implementing role-based security model [24]. Another unique requirement was capturing multiple architecture solutions proposed for a single scenario by different contractors. Each combination of scenario and proposed solution needed to have its own findings attached. Again this requirement has been satisfied by modifying PAKME's repository structure and interface.

AMS's experience of using PAKME has been quite encouraging. During a simulated architecture evaluation project, AMS's evaluators used PAKME as a communication and knowledge sharing mechanism. General scenarios and design options captured in the knowledge base helped them in generating concrete scenarios and understanding proposed solutions. Having a codified quality model provided all evaluators with the same understanding of the quality requirements. Moreover, evaluators found PAKME's templates to capture justification for evaluation decision are very useful. Overall, evaluators and subject matter experts found that the use of an evaluation framework and knowledge management tool brought added rigour to the evaluation process. It is anticipated that the management of evaluation decisions and their justification using PAKME would minimize the need for contacting the evaluator of past projects for explanation.

The modified version of PAKME provides AMS with an effective and efficient mechanism to organise and understand large amount of architecture knowledge. During this trial, AMSs' team have identified several requirements to further enhance PAKME, some of them are mentioned in section 6, however, the current version of PAKME is suitable for capturing and managing several types of architectural knowledge and artefacts of an airborne mission system for evaluating architecture.

AMS is more convinced that an architecture knowledge management tool like PAKME will provide them with several benefits outlined in section 5.2 and help them institutionalising a disciplined evaluation process. In light of new mandated role for DSTO in Defence acquisitions, PAKME will provide AMS with a centralized infrastructure for storing and revisiting evaluation decision quickly and codifying the software architecture evaluation process and practices.

6. Conclusion and Future Work

This research is aimed at improving the effectiveness of architecture-based software engineering through a knowledge management support mechanism. A framework for capturing and using architecture knowledge and a tool, PAKME, to support that framework have been developed. This paper discusses various architectural aspects and features provided by PAKME. This paper also reports on the logistics and our experiences of tailoring and trialling PAKME for evaluating architecture of an aircraft system. During this trial, PAKME and the conceptual framework underpinning it have proven to be adaptable and useful to complex domains like Defence. This has been demonstrated by successfully tailoring and trialling PKAME in a Defence acquisition evaluation setting. Based on the feedback from AMS's evaluators, NICTA is more convinced that its architecture knowledge management framework and tool have the potential to help organisations improve their software architecture processes and build architectural capabilities.

NICTA and AMS have planned further trials of PAKME in future architecture evaluation projects. Based on the current trial, following are some of the planned enhancements to PAKME:

Implementing metrics to measure the usage of the different artefacts of architecture knowledge. Such a feature will provide a feedback loop to improve the type of knowledge captured and features provided.

Improving the speed and accuracy of knowledge retrieval by using the task-based retrieval techniques.

Integrating PAKME with a requirements management tool used in DSTO domain. Such integration will provide an effective mechanism to maintain traceability from requirements to scenarios, to architecture design decisions along with the contextual knowledge underpinning design decisions. Moreover, it will also minimize the duplication of data entry. A similar integration with an architecture modelling/description tool has also been planed.

Acknowledgement – *Several undergraduate students helped us build the PAKME tool. National ICT Australia is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council. The authors would also like to thank the AMS branch members who provided their domain knowledge and expertise to assist in generating the framework and quality attributes used to tailor the PAKME tool to the military mission system domain.*

7. References

[1]. P.N. Robillard, The role of knowledge in software development, *Communication of the ACM*, 1999. **42**(1): pp. 87-92.

[2]. 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.

[3]. 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, *Journal of Systems and Software, Article in the press*, 2006.

[4]. P. Clements, R. Kazman, and M. Klein, Evaluating Software Architectures: Methods and Case Studies. 2002: Addison-Wesley.

[5]. L. Bass and R. Kazman, Architecture-Based Development, *Tech Report CMU/SEI-99-TR-007*, Software Engineering Institute (SEI), Carnegie Mellon University, Pittsburgh, USA, 1999.

[6]. 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.

[7]. 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.

[8]. 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, Standford, United States, 1991.

[9]. J. Tyree and A. Akerman, Architecture Decisions: Demystifying Architecture, *IEEE Software*, 2005. **22**(2): pp. 19-27.

[10]. J. Bosch, Software Architecture: The Next Step, *European Workshop on Software Architecture*, 2004.

[11]. 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.

[12]. 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.

[13]. P. Kruchten, P. Lago, and H.V. Vliet, Building up and Reasoning about Architecture Knowledge, *Proccedings of the 2nd International Conference on Quality of Software Architectures*, 2006.

[14]. M. Ali-Babar, B. Kitchenham, P. Maheshwari, and R. Jeffery, Mining Patterns for Improving Architecting Activities - A Research Program and Preliminary Assessment, *Proceedings of the 9th International conference on Empirical Assessment in Software Engineering*, 2005.

[15]. 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.
[16]. T.R. Gruber 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.
[17]. A.P.J. Jarczyk, P. Loffler, and F.M.S. III, Design Rationale for Software Engineering: A Survey, *Proc. 25th Hawaii Int'l. Conf. on System Sciences*, 1992.
[18]. L.G. Williams and C.U. Smith, PASA: An Architectural Approach to Fixing Software Performance Problems, *Proceedings of the International Conference of the Computer Measurement Group*, 2002.
[19]. G.J.B. Probst. Practical Knowledge Management: A Model That Works. Last accessed on 14th March, 2005, Available from: http://know.unige.ch/publications/Prismartikel.PDF.
[20]. I. Rus and M. Lindvall, Knowledge Management in Software Engineering, *IEEE Software*, 2002. **19**(3): pp. 26-38.
[21]. V.R. Basili and G. Caldiera, Improving Software Quality Reusing Knowledge and Experience, *Sloan Management Review*, 1995. **37**(1): pp. 55-64.
[22]. V.R. Basili, G. Caldiera, and H.D. Rombach, The Experience Factory, in Encyclopedia of Software Engineering, J.J. Marciniak, Editor. 2001, John Wiley & Sons.
[23]. L. Zhu, M. Ali-Babar, and R. Jeffery, Mining Patterns to Support Software Architecture Evaluation, *Proceedings of the 4th Working IEEE/IFIP Conference on Software Architecture*, 2004.
[24]. Hipergate. An open source CRM and Groupware system. Last accessed on 16th March, 2006, Available from: http://www.hipergate.com.
[25]. R. Jeffery and V. Basili, Validating the Tame Resource Data Model, *Proceedings of the10th International Conference on Software Engineering*, 1988.
[26]. 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.
[27]. M. Ali-Babar, B. Kitchenham, and P. Maheshwari, The Value of Architecturally Significant Information Extracted from Patterns: A Controlled Experiment, *Proceedings of the 17th Australian Software Engineering Conference*, 2006.
[28]. M. Ali-Babar, B. Kitchenham, and P. Maheshwari, Assessing the Value of Architectural Information Extracted from Patterns for Architecting, *Proceedings of the 10th International conference on Empirical Assessment in Software Engineering*, 2006.
[29]. 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.
[30]. M.T. Hansen, N. Nohria, and T. Tierney, What's Your Strategy For Managing Knowledge? *Harvard Business Review*, 1999. **March-April**: pp. 106-116.
[31]. K.C. Desouza and J.R. Evaristo, Managing Knowledge in Distributed Projects, *Communication of the ACM*, 2004. **47**(4): pp. 87-91.
[32]. 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.
[33]. L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice. 2 ed. 2003: Addison-Wesley.
[34]. 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.
[35]. 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.
[36]. B. Skuce, Knowledge management in software design: a tool and a trial, *Software Engineering Journal*, Sept. 995: pp. 183-193.
[37]. S. Henninger, Tool Support for Experience-Based Software Development Methologies, *Advances in Computers*, 2003. **59**: pp. 29-82.
[38]. D. Alur, J. Crupi, and D. Malks, Core J2EE Patterns: Best Practices and Design Strategies. 2nd ed. 2003: Sun Microsystem Press.
[39]. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, Pattern-Oriented Software Architecture: A System of Patterns. 1996: John Wiley & Sons.
[40]. L. Bass, M. Klein, and F. Bachmann, Quality Attribute Design Primitives, *Tech Report CMU/SEI-2000-TN-017*, SEI, Carnegie Mellon University, USA, 2000.

[41]. 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.
[42]. P. Clements, et al., Documenting Software Architectures: Views and Beyond. 2002: Addison-Wesley.
[43]. J.L. Kolodner, Improving Human Decision Making through Case-Based Decision Aiding, *AI Magazine*, 1991. **12**(2): pp. 52-68.
[44]. 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.
[45]. Defence Electronic Systems Sector Stragic Plan, C. Department of Defence, Australia, Editor. Feb 2004. pp. 97.
[46]. M. Barbacci, Clements, P., Lattanze, A., Northrop, L., Wood, W., Using the Architecture Tradeoff Analysis Method (ATAM) to Evaluate the Software Architecture for a Product Line of Avionics Systems: A Case Study, *Tech Report CMU/SEI-2003-TN-012*, Carnegie Mellon Software Engineering Institute, July 2003.
[47]. A. Kamel, M. Chandra, and P.G. Sorenson, Building an Experience-Base for Product-line Software Development Process, *ACM*, 2001.
[48]. M.R. Barbacci, P. Clements, A. Lattanze, L. Northrop, and W. Wood, Using the Architecture Tradeoff Analysis Method (ATAM) to Evaluate the Software Architecture for a Product Line of Avionics Systems: A Case Study, *Tech Report CMU/SEI-2003-TN-012*, SEI, Carnegie Mellon University, USA., 2003.