UML Profiles for Design Decisions and Non-Functional Requirements

Liming Zhu[1,2],Ian Gorton[3]
[1]*Empirical Software Engineering Program, National ICT Australia Ltd.*
[2]*School of Computer Science and Engineering, University of New South Wales*
Liming.Zhu@nicta.com.au
[3]*Pacific Northwest National Laboratory, USA*
ian.gorton@pnl.gov

Abstract

A software architecture is composed of a collection of design decisions. Each design decision helps or hinders certain Non-Functional Requirements (NFR). Current software architecture views focus on expressing components and connectors in the system. Design decisions and their relationships with non-functional requirements are often captured in separate design documentation, not explicitly expressed in any views. This disassociation makes architecture comprehension and architecture evolution harder.

In this paper, we propose a UML profile for modeling design decisions and an associated UML profile for modeling non-functional requirements in a generic way. The two UML profiles treat design decisions and non-functional requirements as first-class elements. Modeled design decisions always refer to existing architectural elements and thus maintain traceability between the two. We provide a mechanism for checking consistency over this traceability. An exemplar is given as a way to demonstrate the feasibility of our approach.

1. Introduction

A software architecture is defined as "the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them"[3]. Most Architecture Description Languages (ADL), including graphical views supported by these ADLs, reflect this definition by focusing on expressing components, connectors, properties on them and relationships among them [10]. However, such views of architecture are not suitable for all situations. Firstly, as systems become larger, more decentralized and continue to change, a prescriptive structural architecture has difficulty in supporting flexibility and evolvability [12]. In such situations, an architecture can be expressed as design decisions focusing on quality-centric architectural rules in addition to structural prescription. Secondly, structural descriptions of an architecture alone do not necessarily aid architecture activities which need to capture design decisions and the rationale behind the structural components and connectors. Thus, viewing an architecture directly as a set of NFR/FR-affecting design decisions and treating them as first- class entities has been seen as the logical next-step of software architecture research [4, 17]. This has attracted more fundamental research into design decisions [2, 11]. However, the ability to capture design decisions and their relationship with NFRs in architecture descriptions and views is still lacking. Most of such information is currently captured in separate free-text documentations [5] or content management systems [1]. Such a disconnect has a number of problems:

1) As an architecture evolves, synchronization between semi-formal architecture views and text-based documentations is costly.
2) Switching between architecture views and external documentation for architecture comprehension is inefficient.
3) Architecture decisions are often cross-cutting. One decision relates to multiple architectural elements. Expressing this outside architecture views is difficult.
4) The rigor and friendliness to automated analysis of unstructured textual descriptions is problematic.

In this paper, we propose a new UML profile for modeling design decisions as first-class entities. The profile allows the capturing of design decision using the Object Constraint Language (OCL) [14]. It supports the specification of design rules and constraints by referring to critical participating elements. A consistency checker is developed to check traceability between design decisions and related architectural elements.
The rationale behind each architecture decision is mostly about achieving certain NFRs. This relationship between design decisions and NFRs is modeled using specialized dependency notations in UML. The dependency semantics are modeled after the Goal-oriented Requirement Language (GRL) notation [8]. The

relationships effectively reflects the rationale behind each design decisions.

There exist many attribute-specific, purposely built UML profiles for modeling certain NFRs. These existing profiles can be used directly in our approach. However, we propose a new UML profile for modeling NFRs in a generic but extensible way. Our generic NFR profile only acts as a simple alternative if 1) no appropriate NFR profiles exist or 2) NFRs are mainly captured for documentation purposes rather than formal analysis purposes. The profile is modeled after the six-element framework for non-functional scenarios proposed by the SEI [3].

We consider this work important because:

1) It is an attempt to model cross-cutting design decisions directly in UML views.
2) It semi-formally captures design rationale as relationships between design decisions and NFRs rather than textual explanation.
3) Using UML also enables automated analysis, e.g. consistency checking or NFR coverage checking.

The rest of the paper is organized as follows. Section 2 describes some of the related work in this area and several approaches for expressing cross-cutting features on UML diagrams. Section 3 outlines the design principles behind our approach with some examples. A real world exemplar is used to demonstrate our approach in section 4. We discuss limitations and conclude our paper in section 5.

2. Background and Related work

2.1 UML and OCL

UML is a general purpose modeling language. Extension mechanisms exist for customizing it for new usages. In our approach, we use profiles as the extension mechanism for modeling design decisions. This includes *stereotypes* for classifying model elements and defining new types of model elements, *properties* for specifying the characteristics of a particular model element and *tagged values* for describing keyword-value pairs of model elements, where keywords are attributes. Concepts and constructs in design decisions are mapped to UML using these mechanisms.

Furthermore, we can use OCL to express constraints that specify conditions. These conditions that must be satisfied (or regarded as true) for the model elements can be used to express particular design rules or constraints.

2.2 Expressing design decisions as cross-cutting concerns

One challenging issue for modeling design decisions is about expressing cross-cutting concerns on a design surface, as design decisions usually relate to more than one architectural elements spread across different places and different abstraction levels. There is some work [6] focusing on different ways of expressing pattern usage on top of an existing architecture that is applicable to our approach.

2.2.1 Venn Style annotation

This type of notation for identifying patterns in design diagrams is based on the Venn diagrams [18]. An example of these is shown in Figure 1.

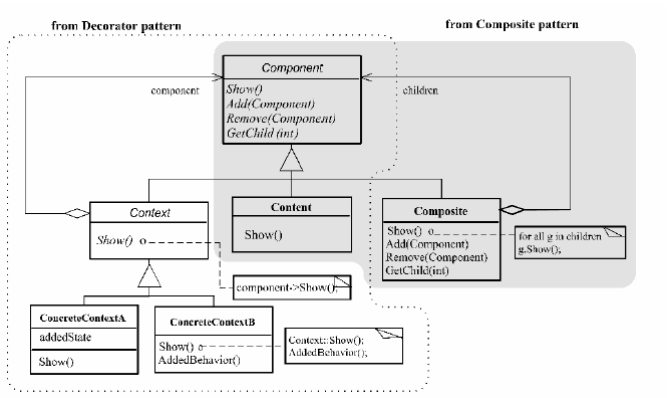

Figure 1. Venn Diagram-Style Annotation (Reproduced from [6])

In the example we can observe that the participants of a cross-cutting entity (pattern, in this case) are identified using the shaded or dotted methods. This notation works fine as long as there are not many patterns or participations of the same element in many patterns. However, the overlapping regions start to get messy and hard to identify. Another major disadvantage of this approach is that it does not show the relationships between participating elements, which is critical to defining design decisions.

2.2.2 UML Collaboration Notation

An improvement over Venn-Style annotation has been proposed to solve some of these problems. The notation used is called parameterized collaboration diagrams [15]. The advantage of using this notation is that the dashed lines (with names) are used to associate the patterns with their participating classes, thus solving the shading problem as shown in Figure 2. However, it also raises other problems such as too many dashed lines will lead to cluttered diagram. More importantly, cross-cutting

information is mixed with design elements, making it hard to identify and potentially polluting design diagrams.

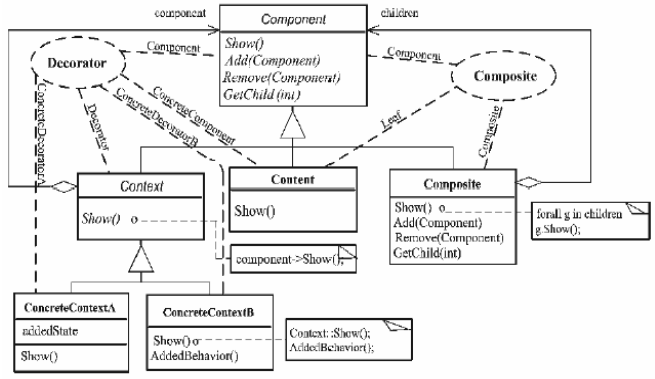

Figure 2. UML Collaboration Notation (Reproduced from [6])

2.2.3 Tagged Annotation

In order to address the cluttering problem, a notation called "tagged annotation" is introduced. This idea makes use of the UML built-in extensibility mechanisms, especially on the tagged values which are used to extend the properties of a model element. For each given class and its attributes and operations, tagged values are created to hold cross-cutting concern related information. In addition, patterns and participant names associated with a class is also placed into a new compartment of the class effectively a first-class entity. An example is shown in Figure 3. The main advantage of this notation is that it gives much better scalability than other notations without losing its readability and captured information. However, information regarding one cross-cutting concern can still spread across the whole diagram.

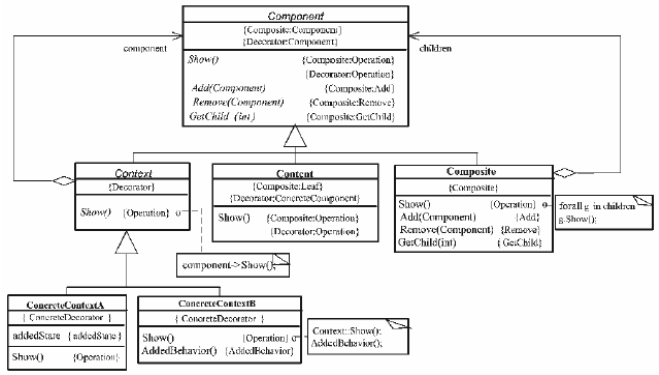

Figure 3. Tagged Annotation with New Compartments (Reproduced from [6])

Our approach follows the tagged annotation approach in spirit but creates a separate entity for design decisions in order to aggregate all related information in one place. We also provide a consistency checker to facilitate analysis across design decision entities and architectural elements.

2.3 Modeling Non-functional Requirements and Design Rationale

Many attribute-specific, purposely built NFR UML profiles exist such as "NoFun" [7] and the OMG performance and scheduling profile [13]. We encourage the reuse of such profiles with our design decision approach. However, these profiles are usually analysis-focused and attribute-specific. In order to express NFRs from a range of quality attributes, multiple such profiles have to be used, which is often overkill for expressing NFRs simply for documenting purposes. For demonstrating the design decision profile, we designed a new generic extensible UML profile for NFR, modeled after the 6-element framework from the SEI [3].

Issues of design decision representation have been addressed in the design rationale modeling work [16]. Design rationales are modeled as first-class entities. However, a design decision is simply viewed as one or more architecture elements. Although these architecture elements implicitly embody certain design decisions, architecture elements alone do not express these decisions clearly. Architecture decisions are much richer as we will see in the next section.

3 UML profiles for design decisions and NFRs

3.1 A UML profile for design decisions

We can identify some important aspects of a design decision [4]:

- Decision: A general design decision can be expressed in OCL, textual formats and any other appropriate domain specific dialects.

- Design rules: These define rules which need to be followed by components within a system. A rule simply describes a particular way of doing something. In the profile, OCL or other dialects can be used to capture design rules. Rules are important since they provide a more flexible way of regulating architecture quality properties than structural prescriptions in certain situations such as Ultra-Large-Scale systems [12].

- Design constraints: Other than design rules, a design decision may contain constraints which specify what the system may or may not do. Similar to above, OCL will be used as an expression syntax for design constraints.

- Participating elements: Participating elements are architectural elements to which a design decision refers to. It essentially extracts the context part of the decision/decision rules/decision constraints expression in OCL.

• Rationale: Rationales can be captured descriptively in a separate tag along the relationships to NFRs expressed using UML extension mechanisms.

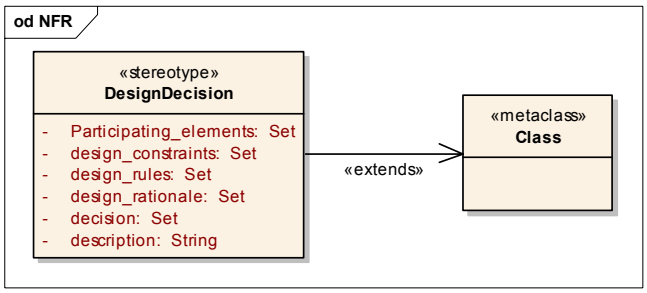

Figure 4. UML profile for design decisions

Figure 4 shows the stereotype *DesignDecision* along with its attributes as described early. It is not necessary for modelers to fill in all the tag values for all the design decision classes except for the decision one. Additional meta-attributes can be added to this design decision meta-model, in order to suit specific needs from architects.

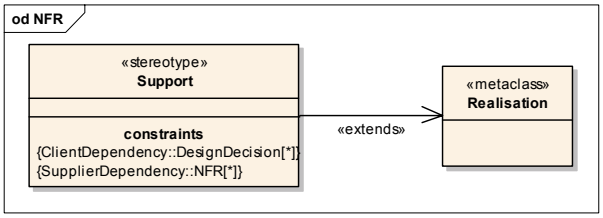

Figure 5. Stereotype 'Support'

Figure 5 shows one of the stereotypes for modeling relationships between decision decisions and NFRs. A design decision can support/break/help/hurt NFR. One decision can affect multiple NFRs.

3.2 A UML profile for NFRs

According to [3], a NFR can be expressed in scenarios. A scenario could be expressed in forms such as unstructured text or template-based formats such as the 6-element framework: stimulus, source of stimulus, environment, artifact, response and response measures.

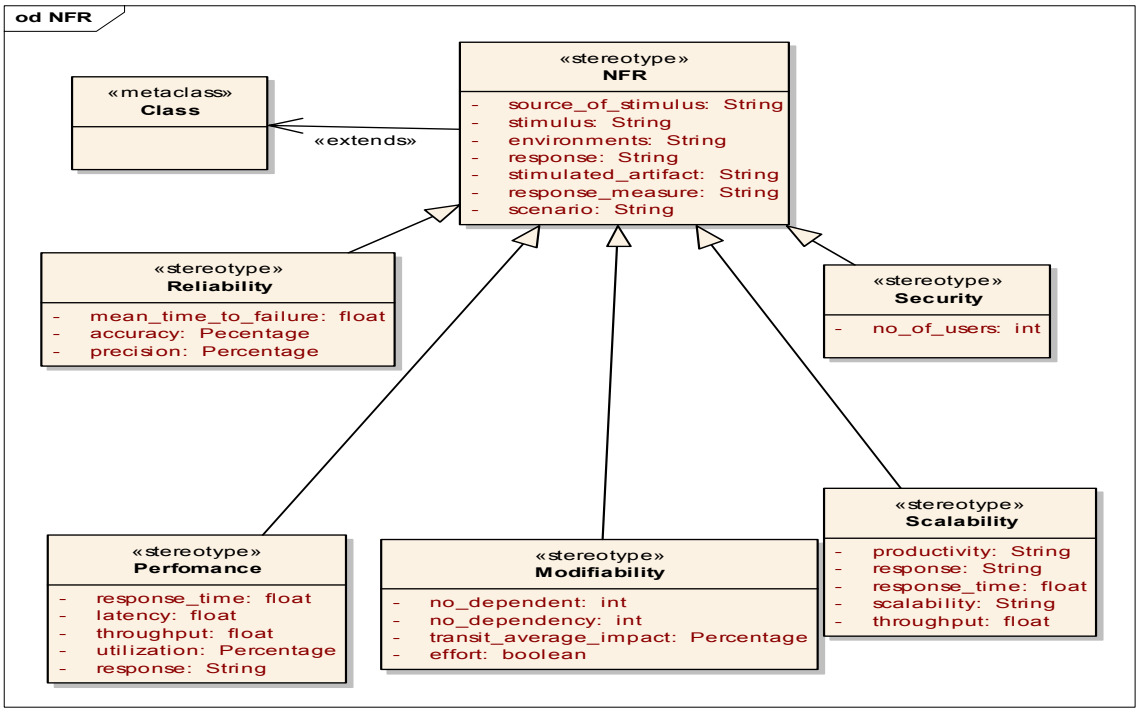

Figure 6. UML profile for NFR

We capture this in an extensible UML profile as shown in Figure 6. The super-class NFR stereotype captures the six elements along with a description. Attribute-specific stereotypes inherit and can be further extended to include attribute specific entities, especially response-measures. For example, a modifiability NFR includes response measures such as number of dependents, transit average impact and effort.

3.3. Profile usage

The profiles are expected to be used by architects and developers. Relevant stakeholders who would like to analyze these models to meet their business requirements are also expected to use this profile. These profiles are specified in XMI-compatible XML format. Some UML modeling tools, such as Enterprise Architect can import this XML file and allow the modeler to use them in their modeling environment. Modelers may add additional elements to the profile to suit their requirements in any meta-modeling environment. Other than adding additional elements on the meta-level models, modelers may also add optional tags on the instance level to best suit their requirements. Separate packages for each type of quality attributes are recommended for better readability.

4. Case Study

We use a real world generic caching service [9] as a case study to demonstrate the feasibility of the profiles.

4.1 Generic Caching Service

Many applications need to support tens of thousands of concurrent users. In order to gain high quality levels of scalability and performance, resource caching is a mechanism often used. A number of NFRs related to performance are shown in Figure 7:

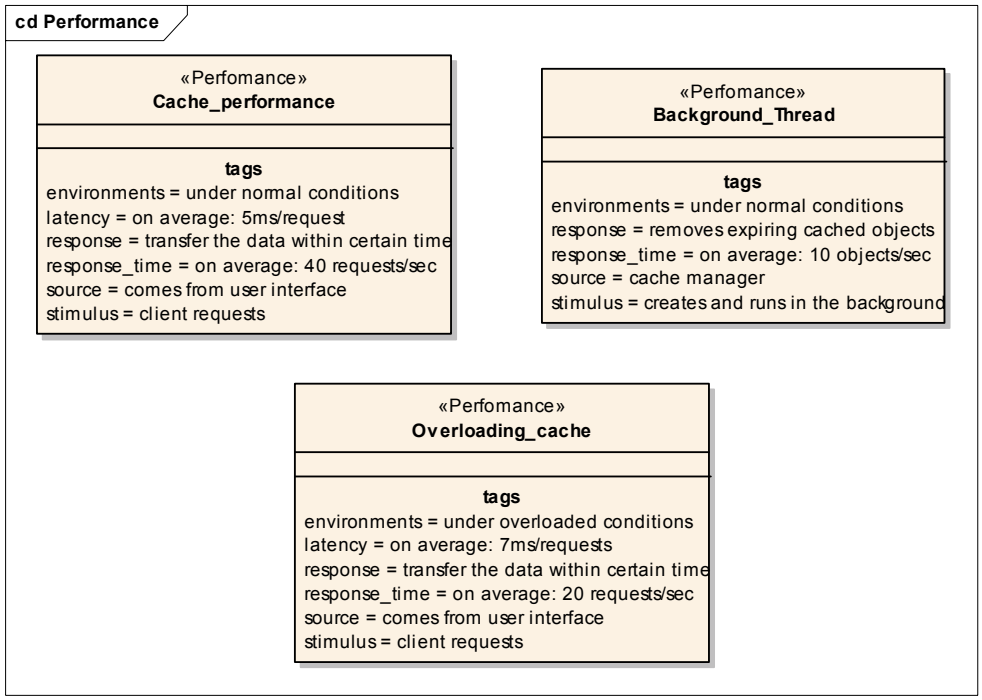

Figure 7. Performance NFRs

Other NFRs, such as reliability and scalability are shown along with the design decisions in Figure 8 and Figure 9.

In the cache manager class, there are three decisions. The first decision refers to the size of the cache, in this example it's 500. The second decision refers to one of the attributes within the cache manager class which specifies whether the LRU algorithm has been used. The third decision refers to the setting of the priority of the thread which runs in the background for purging expired objects. Most of the decisions are also expressed in a free text form, for example, "placeholder for enhancing caching service through a new algorithm" which corresponds to the decision "cache.CacheManager, newAlgorithmLRU = true". The main reason for this is that the limitations of OCL.

In Figure 9, the design decision class "supports" one of the performance NFR and "breaks" another performance NFR depending on the conditions expressed within the NFR. The design decision "supports" the

"cache_performance" NFR under normal conditions but "breaks" it under overloaded conditions.

Some of the design decisions are listed below:

- Methods *putObject()* and *getObject()* have been implemented in the *CacheManager* class in order to place/extract objects to/from the cache.
- Cached objects can determine when they expire through the *isExpired()* method.
- A background thread that runs under low priority is implemented by the attribute *threadCleanerUpper* in the *CacheManager* to satisfy this requirement.
- Code can be added to the *threadCleanUpper* to search out the LRU cached objects, and hence this service can be enhanced later through the use of the LRU algorithms for purging cached objects.

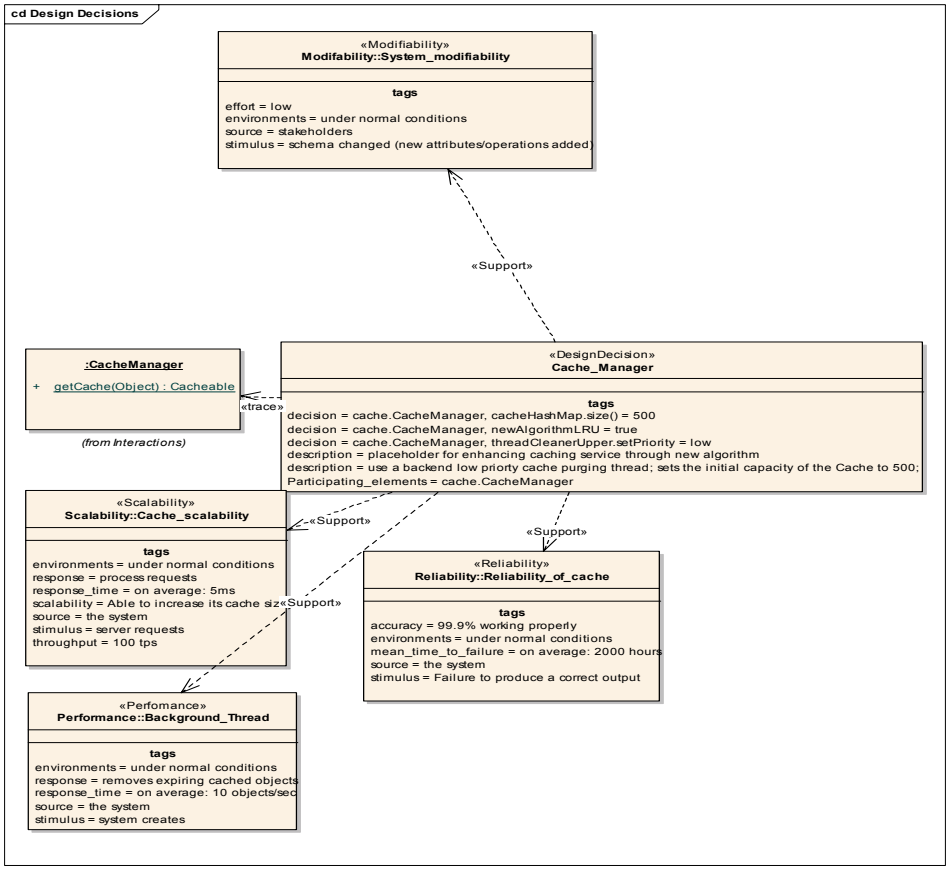

Figure 8. Design decisions related to CacheManager

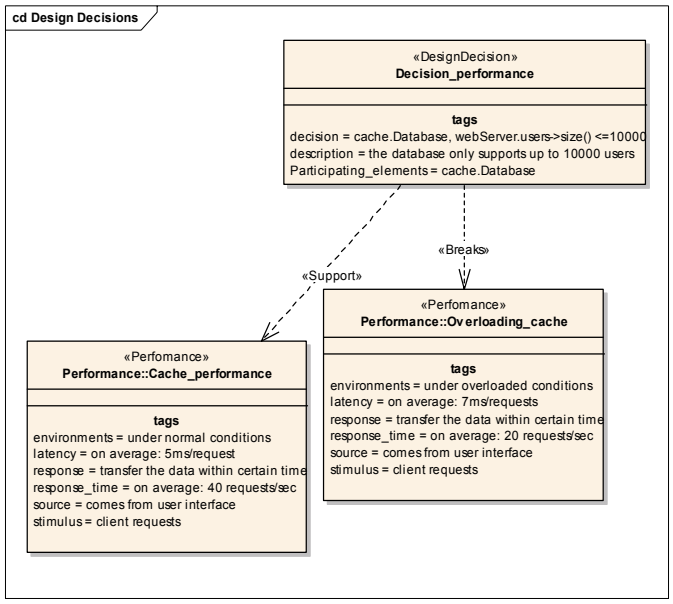

Figure 9 Decision related to database performance

5. Conclusion

In this paper, we propose a UML profile for modeling design decisions and an associated UML profile for modeling non-functional requirements in a generic way. The two UML profiles treat design decisions and non-functional requirements as first-class elements. Modeled design decisions always refer to existing architectural elements and thus maintain traceability between the two. We are currently validating the profile in two ways:

1) We are applying the approach in real projects to validate its effectiveness.

2) We are seeking expert opinions to evaluate its expressiveness, intuitiveness, clarity and scalability.

5. Acknowledgements

National ICT Australia is funded by the Australian Government's Department of Communications, Information Technology, and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Research Centre of Excellence programs.

[1] M. A. Babar, I. Gorton, and B. Kitchenham, "A Framework for Supporting Architecture Knowledge and Rationale Management," in *Rationale Management in Software Engineering*, 2006.

[2] F. Bachmann, L. Bass, and M. Klein, "Deriving Architectural Tactics: A Step Toward Methodical Architectural Design," Software Engineering Institute CMU/SEI-2003-TR-004, 2003.

[3] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2 ed.: Addison-Wesley, 2003.

[4] J. Bosch, "Software architecture: the next step," in *1st European Workshop on Software Architecture (EWSA)*, 2004.

[5] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, *Documenting software architectures : views and beyond*. Boston: Addison-Wesley, 2003.

[6] J. Dong, "UML Extensions for Design Pattern Compositions," *Journal of Object Technology,* vol. 1, pp. 151-163, 2002.

[7] X. Franch, "Systematic Formulation of Non-Functional Characteristics of Software," in *3rd International Conference on Requirements Engineering (ICRE)*, 1998.

[8] L. Liu and E. Yu, "Designing Information Systems in Social Context: A Goal and Scenario Modelling Approach," *Information Systems,* vol. 29, 2003.

[9] J. Lurie, "Develop a Generic Caching Service to Improve Performance, http://java.sun.com/developer/technicalArticles/ALT/cachingservices/," 2002.

[10] N. Medvidovic and R. N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages," *IEEE Transactions on Software Engineering,* vol. 26, pp. 70-93, Jan, 2000.

[11] M. Moore, R. Kazman, M. Klein, and J. Asundi, "Quantifying the value of architecture design decisions: lessons from the field," in *25th International Conference on Software Engineering (ICSE)*, 2003.

[12] L. Northrop, R. Kazman, M. Klein, D. Schmidt, K. Wallnau, and K. Sullivan, "Ultra-Large Scale Systems: The Software Challenge of the Future," 2006.

[13] OMG, "UML Profile for Schedulability, Performance and Time, v1.0."

[14] OMG, "UML 2.0 Object Constraint Language (OCL) Specification," 2004.

[15] J. Rumbaugh, I. Jacobson, and G. Booch, *The unified modeling language reference manual*, 2nd ed. Boston: Addison-Wesley, 2005.

[16] A. Tang and J. Han, "Architecture Rationalization: A Methodology for Architecture Verifiability, Traceability and Completeness," in *12th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS)*, 2005, pp. 135-144.

[17] J. Tyree and A. Akerman, "Architecture Decisions: Demystifying Architecture," in IEEE Software. vol. 22, 2005, pp. 19-27.

[18] J. Vlissides, "Notation, Notation, Notation." " in C++ Report. vol. April, 1998, pp. 48-51.