Scenarios, Quality Attributes, and Patterns: Capturing and Using their Synergistic Relationships for Product Line Architectures

Muhammad Ali Babar
National ICT Australia Ltd. and University of New South Wales, Australia
malibaba@cse.unsw.edu.au

Abstract

Typically, architectural choices determine the achievement of desired goals (such as reusability and maintainability) of product line software development. Several methods have been proposed to design and analyze product line architectures with respect to desired quality attributes. Most of these methods encourage the use of architectural patterns to develop architectures with known characteristics and apply scenarios to evaluate those architectures for desired quality attributes. We observe an increased awareness of the links that exist among scenarios, quality attributes, and patterns. However, there are very few attempts to systematically capture and suitably document such synergistic relationships to support architecture design and evaluation. This paper presents our thoughts on exploiting the above-mentioned synergy. It also proposes some techniques of improving the product line architecture design and evaluation process by identifying and capturing architecturally significant information from architectural patterns.

1. Introduction

Software architecture (SA) of a product family constrains the achievement of various quality attributes (such as reusability, performance, security, maintainability and usability) [1]. A number of methods, such as Architecture Tradeoff Analysis Method (ATAM) [2], Quality Attribute-oriented Software Architecture design method (QASAR) [3] and Quality-driven Architecture Design and Analysis (QADA) [4], have been developed to pay significant attention to quality related issues at the SA level. These approaches heavily depend on architectural styles and patterns to design candidate architectures and on scenarios and associated reasoning frameworks to evaluate SAs for desired quality attributes.

A quality attribute is a non-functional requirement of a system, e.g., reliability, modifiability and so forth. Scenarios are widely used in product line software engineering: abstract scenarios to capture behavioural requirements and quality-sensitive scenarios to specify architecturally significant quality attributes.

Patterns[1] embody proven solutions to recurring problems by capturing concentrated wisdom of many experienced practitioners. Patterns are a vital means of designing SAs of large complex systems. One of the major objectives of using patterns is to develop a software system with predictable non-functional properties [1]. Each pattern helps achieve one or more quality attribute and may hinder others. In product line development, an architect designs a desirable SA by composing a number of architectural styles and patterns.

Scenarios are used to characterize quality goals and patterns are used to achieve those goals. Moreover, scenarios are used to reason about the design decisions during SA design stage and to evaluate that SA with respect to target quality requirements. Recently, there have been attempts to codify the links that exist among scenarios, quality attributes, and patterns [5, 6].

However, there has been little research on systematically studying, identifying, and documenting the synergistic relationships that informally exist in each pattern's description. There are different formats of describing patterns. However, each type of format uses at least four major sections to document a particular pattern, i.e. context, problem, solution and quality consequences. We have observed that each documented pattern has several scenarios, though informally described, which specify the quality attributes obtained by the pattern and associated quality consequences. We believe that this invaluable information that should be systematically captured and documented as reusable artifacts to improve product line SA design and evaluation processes.

Software architects can greatly benefit from the abstract scenarios distilled from the patterns during SA design. These scenarios can be used to precisely articulate the quality requirements and ensure the quality requirements are complete. These general scenarios can also help an architect identify suitable architectural mechanisms for architectural drivers. Moreover, these scenarios can also support to apply more rigorous reasoning about the design decisions with respect to the desired quality attributes. From SA evaluation point of view, if the abstract scenarios that characterize the quality attributes satisfied by the patterns used in the SA are available, it will improve SA evaluation and reduce

[1] By patterns we mean architecture styles or patterns.

Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04)
1530-1362/04 $ 20.00 IEEE

the time and resources required to gather scenarios from scratch for each evaluation effort.

There is very little use of the architectural information, embedded in patterns, during SA design and evaluation processes. We see two reasons for this: 1) there is little guidance on how to extract the architectural information from patterns; 2) existing format of pattern documentation is not appropriate to explicitly codify the relationships of scenario, quality attributes, and patterns. In [7], we demonstrated how to mine patterns for architectural information. To address the second issue, we propose a framework of two templates to document the architectural information extracted from patterns in a manner that facilitates the use of this information during SA design and evaluation activities.

2. Background and Motivation

Scenarios have been found quite effective and useful for precisely specifying quality attributes. Several approaches use scenarios to encourage disciplined thinking during SA design and evaluation activities [4, 8]. Scenarios are considered very flexible as they can be used for systematically reasoning about or evaluating most of the quality attributes. For example, we can use scenarios that represent failure to reason about the appropriate architectural mechanisms to satisfy availability and reliability requirements, scenarios that represent change requests can be used to examine modifiability, scenarios that represent threats can help select appropriate security tactics, or scenarios that represent ease of use can support to assess usability. Moreover, scenarios are normally very concrete, enabling the user to understand their detailed effect [9].

A quality attribute is a non-functional requirement of a software system, e.g., security, flexibility, changeability, portability and so forth. According to [10], software quality is the degree to which the software possesses a desired combination of attributes. Quality attributes of large software intensive systems are usually determined by the system's SA. It is widely recognized that SA of product family constraints the achievement of desired set of quality attributes (i.e. variability, flexibility etc.) [1]. Since SA plays a vital role in achieving system wide quality attributes, it is very important to appropriately focus on quality related issues during SA design and rigorously assess the capability of the designed SA with respect to the desired quality attributes as early as possible.

Table 1. A template to document architecturally significant information found in a pattern

Pattern Name: *Name of the software pattern*		**Pattern Type:** *Architecture, design, or style*	
Brief description		*A brief description of the pattern.*	
Context		*The situation for which the pattern is recommended.*	
Problem description		*What types of problem the pattern is supposed to address?*	
Suggested solution		*What is the solution suggested by the pattern to address the problem?*	
Forces		*Factors affecting the problem & solution. Justification for using pattern.*	
Available tactics		*What tactics are used by the pattern to implement the solution?*	
Affected Attributes		**Positively**	**Negatively**
		Attributed supported	*Attributed hindered*
Supported general scenarios	S1		
	S2		
	S..n		
Usage examples	*Some known examples of the usage of the pattern to solve the problems.*		

A pattern is known solution to a recurring problem in a particular context. Patterns provide a mechanism of documenting and reusing the design knowledge accumulated by experienced practitioners [11]. Software architecture for product family is usually composed of patterns. One of the main goals of using patterns is to design a SA with known quality attributes [12]. Each pattern either supports or inhibits one or more quality attributes. There are several formats of documenting patterns, however, most of the patterns' documentation includes at least four sections: context, problem, solution and quality consequences. Each pattern's description has large amount of architecturally significant information, e.g. scenarios, quality attributed supported or hindered, forces, tactics and so on. The scenarios, informally embedded in a pattern description, can be used to characterize the quality attributes achieved by the pattern.

This background discussion brings us to the point where we can clearly see the links among scenarios, quality attributes, and software patterns. Recently, there has been some effort to make these relationships explicit and codify this knowledge to support SA design and evaluation activities [5, 6]. However, these approaches do not attempt to systematically capture and document the synergistic relationships among scenarios, quality attributes, and patterns usually informally described within a pattern's documentation. We claim a disciplined approach to extract and document such relationships can help the software engineers to fully exploit these relationships in order to achieve desired combination of various quality attributes.

3. A Proposal

We have demonstrated that software patterns can be mined to extract architecturally important information [7]. However, the extracted information must also be documented in such a format, which makes it readily useable during SA design and evaluation activities.

We propose a framework of two templates to document architectural information (i.e. general scenarios, quality attributes, tactics, usage examples and so on) to support SA design and evaluation processes. We also provide a simple process of identifying and extracting the architectural information from patterns.

Table 1 presents the first template, which captures the information extracted from patterns. This template is more appropriate during product line architecture design stage where abstract scenarios are used to characterize required quality attributes and suitable patterns are chosen. Table 2 presents the second template, which is aimed at documenting the architectural information found in patterns for SA evaluation, which needs concrete scenarios. Scenario-based approaches mainly gather scenarios from stakeholders. We argue that many of the concrete scenarios can be gathered by concretizing the abstract scenarios extracted from patterns. We believe that concrete scenarios and other architectural information documented using template 2 can greatly improve the SA assessment process as architect can confidently tell the stakeholders that scenarios extracted from the patterns have already been considered because of using that particular pattern in that architecture.

Table 2: A template to document architectural information for SA evaluation process

Project Name: *Which project needs this scenario?* **Project domain:** *Domain of the project*		**Date:** *When was proposed?* **Scenarios No:** *Serial number assigned to the scenario*
Business goals	*Which business goals does this scenario achieve?*	
Stakeholders	*Which class of the stakeholders did suggest this scenario?*	
Attributes	*Which quality attributes are required by this scenario?*	
Description	*A brief description of the scenario.*	
Concrete scenario	**Stimulus**	*A condition that needs to be considered when it arrives at a system.*
	Context	*A system's condition when a stimulus occurs, e.g, overloaded, running etc.*
	Response	*A measurable action that needs to be undertaken after the arrival of the stimulus*
	Complexity	*How complex is this scenario to realize? (affect on macro or micro architecture)*
	Priority	*How much important is this scenario?*
Pattern/Style	*Name of the architectural pattern or style that can support this scenario.*	
Design tactics	*What are the design tactics used by the pattern/style to support the scenarios?*	
Design rational	*What are reasons to use the patterns/tactics? How does the pattern provide the desired quality attributes?*	

We claim that these templates make the relationships among scenarios, quality attributes, and patterns explicit. Moreover, the proposed templates also capture one of the most important parts of a pattern description, forces. The forces of a pattern describe the factors whose clash causes the problem that the pattern attempts to solve by resolving the clashes among those factors. This part also captures tradeoff and considerations of designers. The forces of a pattern are usually described implicitly in most of the pattern documentation styles. Recently, software engineering community has started paying appropriate attention to the forces of a pattern in an attempt to fully understand the problem and solution described in a pattern [6, 13].

4. An Example

In this section, we demonstrate how to use the proposed templates to document the information extracted from a pattern. We also describe a simple process that we usually follow to identify and extract relevant information. We have stated earlier that usually a pattern's documentation consists of four sections: context, problem, solution, and quality consequences.

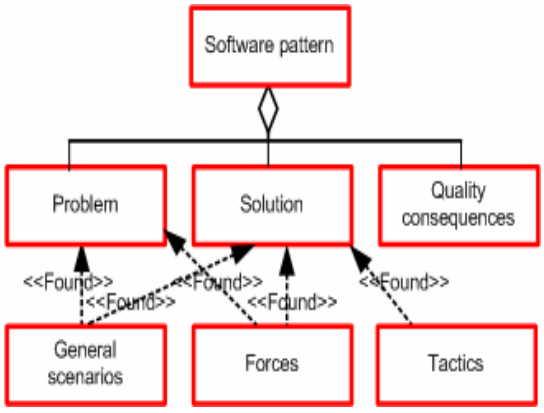

Figure 1. where to find relevant information

Figure 1 presents a diagrammatic guide to spotting architecturally significant information from a pattern description. We have found that scenarios are mostly found in problem and solution sections, while forces can

also be found in these sections. However, there are some patterns description styles that use separate sections for forces. The quality attributes affected (positively or negatively) by the patterns are described in the quality consequence section.

We have followed following simple steps to identify and extract architectural information:

- Select software pattern
- Understand pattern documentation format
- Explore different parts for information
- Capture each type of information separately
- Structure & document information in a readily useable templates

To demonstrate the process of extracting general scenarios and other pieces of information, we have selected "Reflection" pattern [14], which is an architectural pattern. This pattern provides a means of dynamically changing structure and behaviour of software without affecting the key design abstractions in response to evolving technology and requirements. This pattern splits an application into two parts. A meta level provides information about the selected properties of the system and makes the software self-aware. A base level part includes the application logic [14].

We have used Table 1 of our proposed framework of documenting architectural information. Table 3 shows the extracted information from the Reflection pattern.

Table 3. An example of architectural information extracted from a pattern

Pattern Name: Reflection		**Pattern Type:** Architectural Pattern
Brief description	This pattern supports dynamic changes in the structure and the behaviour of software systems.	
Context	Building systems that support changeability.	
Problem description	Changing technology and requirements needs an architecture that is open to modification.	
Suggested solution	Make selected aspects of a software's structure and behaviour accessible for adaptation.	
Forces	Systems can be easily modified in response to changing technology and requirements. Selected aspects of the software can be adaptable and changeable. Split the architecture into two parts.	
Available tactics	Split the architecture into mea level and base level and allow changes using metaobject protocols.	
Affected Attributes	**Positively**	**Negatively**
	Adaptability, Changeability	Performance, efficiency, language independence
Supported Abstract scenarios	S1	A new feature request shall be accommodated without affecting the key design abstraction.
	S2	A change in the environment necessitates changes in the function call mechanism.
Examples of usage	OLE 2.0, Car-dealer system NEDIS, MIP	

5. Discussion and Future Work

This paper presents our idea to improve the product line software architecture design and evaluation process by exploiting the synergistic relationships of scenarios, quality attributes, and patterns. We have demonstrated that patterns' documentations incorporate significant amount of architectural information that should be systematically extracted [7]. However, the extracted information can only be useful if it is documented in a manner that makes it readily useable during the architecture design and assessment activities by making the relationships of scenarios, quality attributes, and patterns explicit. We have proposed a framework of two templates that can effectively capture the extracted information and promote it efficient use. We have shown an example of using one of the proposed patterns and described a simple process that can be followed to extract and document the architectural information.

The main goal of our research is to improve the process of SA design and evaluation by providing architectural information in a format that can support design decisions with an informed knowledge of the consequences of those decisions. More specifically, we intend to minimize the time, resources and skill level required to effectively and efficiently design and analyze a SA for product family. We believe that developing a systematic approach to identify, efficiently extracting, and explicitly documenting the relationships of scenarios, quality attributes, and patterns is one of the most important steps towards that goal. We plan to use experimentation to assess our claim of improving SA design and evaluation process by distilling SA sensitive information from patterns and usefulness of the proposed framework to document the information.

6. References

[1] Bass, L., et al., "Software Architecture in Practice", 2 ed. 2003: Addison-Wesley.

[2] Clements, P., et al., "Evaluating Software Architectures: Methods and Case Studies". 2002: Addison-Wesley.

[3] Bosch, J., "Design & Use of Software Architectures: Adopting and evolving a product-line approach". 2000: Addison-Wesley.

[4] Matinlassi, M., et al., "Quality-driven architecture design and quality analysis method: A revolutionary initiation approach to a product line architecture," Tech. Report VTT Technical Research Centre of Finland, Espoo, 2002

[5] Bass, L. and B.E. John, "Linking usability to software architecture patterns through general scenarios," *Journal of Systems and Software*, 2003. **66**(3): p. 187-197.

[6] Gross, D. and E. Yu, "From Non-Functional Requirements to Design through Patterns," *Proc. of the 6th Int'l Workshop on Requirements Engineering Foundation for Software Quality*. 2000. Sweden.

[7] Zhu, L., et al., "Mining Patterns to Support Software Architecture Evaluation," *Proc. of the 4th Working IEEE/IFIP Conference on Software Architecture*. 2004.

[8] Thiel, S., "On the Definition of a Framework for an Architecting Process Supporting Product Familiy Development," *Proc. of the 4th Int'l Workshop on Product Family Engineering*. 2001. Bilbao, Spain.
[9] Lassing, N., et al., "How Well can we Predict Changes at Architecture Design Time?," *Journal of Systems and Software*, 2003. **65**(2).
[10] "IEEE Standard 1061-1992, Standard for Software Quality Metrics Methodology". 1992, New York: Institute of Electrical and Electronic Engineers.
[11] Gamma, E., et al., "Design Patterns-Elements of Reusable Object-Oriented Software". 1995, Reading, MA: Addison-Wesley.
[12] Buschmann, F., *Pattern-oriented software architecture : a system of patterns*. 1996, Wiley: Chichester ; New York. p. xvi, 457 p.
[13] John, B.E., et al., "Bringing Usability Concerns to the Design of Software Architecture," *Proc. of the 9th IFIP Working Conference on Engineering for Human-Computer Interaction*. 2004. Hamburg, Germany.
[14] Buschmann, F., et al., "Pattern-Oriented Software Architecture: A System of Patterns". 1996: John Wiley & Sons.