Mining Patterns to Support Software Architecture Evaluation

Liming Zhu, Muhammad Ali Babar, Ross Jeffery
National ICT Australia Ltd. and University of New South Wales, Australia
{limingz, malibaba, rossj}@cse.unsw.edu.au

Abstract

In this paper, we present an approach to improve the software architecture evaluation process by systematically extracting and appropriately documenting architecturally significant information from software architecture and design patterns; we are interested in only two pieces of information found in software patterns: general scenarios and architectural tactics. General scenarios distilled from patterns not only assist stakeholders in developing concrete scenarios during a scenario-based architecture evaluation, but can also help an architect select and calibrate a quality attribute reasoning framework. Architectural tactics in patterns are used as a means of manipulating independent parameters in the reasoning framework to achieve the desired quality. Moreover, we believe if we use general scenarios and tactics extracted from patterns in an architectural evaluation, the results of that evaluation can be used as an evidence to validate the pattern's claim with respect to the quality attributes. We demonstrate our approach by using EJB architecture usage patterns. We contend that this approach can be used to analyze and validate any architecture pattern.

1. Introduction

Quality is one of the most important issues in software development. The idea of predicting the quality of a software intensive system from a high level design description originated in Parnas's seminal work on software modularization [1], and has recently emerged as an important quality assurance technique known as software architecture (SA) evaluation. It has been shown that SA constrains the achievement of various quality attributes (such as performance, security, maintainability and usability) in a software intensive system [2]. That is why a number of formal and systematic efforts are concentrated on ensuring that the quality issues are addressed at the SA level. The principle objective of evaluating a SA is to assess the potential of the chosen architecture to deliver a system capable of fulfilling required quality requirements and to identify potential risks [3]. A number of methods, such as Architecture Tradeoff Analysis Method (ATAM) [4], Software Architecture Analysis Method (SAAM) [5] and Architecture-Level Maintainability Analysis (ALMA) [6], have been developed to evaluate the quality related issues at the SA level. Most of these methods are structurally same but there are a number of differences among their activities and techniques [7]. The accuracy of the results of these methods is largely dependent on the quality of the scenarios used for the SA evaluation as these are scenario-based methods [8]. An appropriate Reasoning Framework for a desired quality attribute is also an important element of these methods. The main sources of quality attribute sensitive scenarios and reasoning frameworks are the problem domain, quality requirements, existing quality models and stakeholders [8, 9]. It has been claimed that architectural patterns are another important source of quality attribute sensitive scenarios [10].

Recently, many large and complex software systems have been developed using architectural and design patterns. One of the major purposes of using patterns is to develop software systems that are expected to provide the desired level of quality attributes [9]. Patterns are documented in a variation of Alexander or Gang Of Four (GOF) styles [11, 12], which require the inclusion of problem, solution, quality consequences parts. We observed that quality attribute sensitive scenarios and other architectural related information are embedded in a pattern description. For example, the problem part contains the scenarios addressed by the patterns, architectural tactics and design decisions can be found in the solution part, and quality attribute analysis is provided in the quality consequences section. We believe that this invaluable information can be extracted and systematically documented to support the SA evaluation process.

Another important issue this paper examines is the lack of rigorous validation of the published patterns. Though patterns have been very popular since early 90s, they have been often criticized for lack of systematic and rigorous validation of their benefits [13]. The patterns community has proposed a few solutions to address this issue, e.g., validation through experiments [13], or in anecdotal form from practitioners [14]. However, successful design with patterns largely relies on the experience of the patterns' user.

Since the benefits a pattern claims to offer are achieved by the tactics used in that pattern, we believe that one can rigorously validate a pattern by identifying the independent parameter a pattern's tactic is manipulating and using an appropriate reasoning framework. In order to accumulate the evidence from architecture evaluation for pattern validation, we need to guarantee at least two things:

1) The concrete scenarios we are using in the evaluation must be instances of the general scenarios extracted from the pattern, to ensure that we evaluate the pattern against the problem that pattern claims to solve.

2) The parameters that the pattern's tactics are manipulating must be part of the independent parameters of the quality-attribute reasoning framework, to ensure that the tactic is achieving the benefit the pattern claims to provide.

This paper shows that the information we have extracted from patterns satisfies these two conditions.

This paper makes two significant contributions to architecture evaluation discipline. First, it presents our approach to distill quality attribute sensitive scenarios, architectural tactics and other architecture related information from patterns to support the scenario-based SA evaluation. Second, the paper presents a way of rigorously validating a pattern's benefits.

We begin by discussing the importance of SA evaluation for quality attributes and the role of general scenarios and reasoning frameworks in performing architectural evaluation. We then present our assertion that architectural patterns are an important source of quality attribute sensitive scenarios and other architectural information. We demonstrate our approach by extracting a number of pieces of architecturally vital information from well know software patterns. We illustrate the practical applicability of our approach with an example. We close by mentioning the limitations of this work and identifying areas of future work.

2. Background and Motivation

2.1 Quality attributes and software architecture evaluation

A quality attribute is a non-functional requirement of a software system, e.g., reliability, modifiability, performance, usability and so forth. According to [15], software quality is the degree to which the software possesses a desired combination of attributes. There are a number of classifications of quality attributes. In [16], McCall listed a number of classifications of quality attributes developed by a number of software engineering researchers including himself. A later classification of software quality is provided in [17].

Quality attributes of large software intensive systems are usually determined by the system's SA. During recent years, it has been widely recognized that quality attributes (such as maintainability, reliability, usability, performance, flexibility etc.) of complex software intensive systems depend more on the overall SA of such systems than on other factors such as platform and framework selection, language choice, detailed design decisions, algorithms, data structures, and so forth [2]. Since SA plays a vital role in achieving system wide quality attributes, it is very important to evaluate a system's architecture with regard to desired quality requirements as early as possible. The principle objective of SA evaluation is to assess the potential of the chosen architecture to deliver a system capable of fulfilling required quality requirements and to identify any potential risks [3]. Additionally, it is quicker and less expensive to detect and fix design errors during the initial stages of the software development.

A number of methods and techniques have been applied to ensure that the quality concerns are addressed at the architecture level. Scenario-based SA evaluation methods such as ATAM, SAAM, and ALMA, are considered relatively mature and established as they have been widely applied and more rigorously validated in various domains [7].

Figure 1 shows a generic process of scenario-based SA evaluation. The desired quality attributes, their associated reasoning frameworks and relevant concrete scenarios are important inputs, an evaluation report is the output.

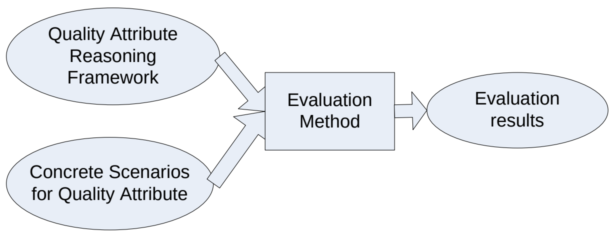
Quality Attribute Reasoning Framework
Concrete Scenarios for Quality Attribute
Evaluation Method
Evaluation results

Figure 1: Important inputs and outputs of software architecture evaluation process

2.2 Using scenarios for software architecture evaluation

Abowd et. al. [18] proposed two broad categories of SA evaluation: questioning techniques, which focus on qualitative analysis; and measurement approaches, which are quantitative in nature. The former category includes techniques like scenarios, questionnaires, and checklists. The latter category consists of metrics and simulation. Most of the mature and established architectural evaluation methods are scenario-based. Scenarios have been used for a long time in several areas of different disciplines (military and business strategy, decision making,). The software engineering community started using scenarios in user-interface engineering, requirements elicitation, performance modelling and more recently in SA evaluation [19] .

Scenarios are quite effective for SA evaluation because they are very flexible; scenarios are used for evaluating most of the quality attributes. For example, we can use scenarios that represent failure to examine availability and reliability, scenarios that represent change requests to analyze modifiability, scenarios that represent threats to analyze security, or scenarios that represent ease of use to analyze usability. Moreover, scenarios are normally very concrete, enabling the user to understand their detailed effect [20].

The SA community has developed different frameworks for eliciting, structuring and classifying scenarios. For example, Nico Lassing et. al. [21] proposed a two dimensional framework to elicit change scenarios, Rick Kazman et. al. [22] proposed a generic 3 dimensional matrix to elicit and document scenarios. Len Bass et. al. [2] provided a six elements framework to structure scenarios. Scenarios used in architecture evaluation are also classified into various categories, e.g., direct scenarios, indirect scenarios, complex scenarios, use case scenarios, growth scenarios, exploratory scenarios and so forth [2, 5, 21].

Elements	**Brief Description**
Stimulus	A condition that needs to be considered when it arrives at a system
Response	The activity undertaken after the arrival of the stimulus
Source of Stimulus	An entity (human, system, or any actuator) that generates the stimulus
Environment	A system's condition when a stimulus occurs, e.g, overloaded, running etc.
Stimulated Artifact	Some artifact that is stimulated; may be the whole system or a part of it.
Response measure	The response to the stimulus should be measurable in some fashion so that the requirement can be tested.

Table 1: Six elements scenario generation framework [9]

We will use the scenario generation framework shown in Table 1 to structure the scenarios distilled from software patterns. This framework provides a relatively rigorous and systematic approach to capture and document general scenarios, which can be used to develop concrete scenarios and to select an appropriate reasoning framework to evaluate SA.

2.3 Role of a reasoning framework in software architecture evaluation

Most of the scenario-based SA evaluation methods use informal reasoning (such as impact analysis or walkthroughs) for analyzing the design decisions with respect to the desired quality attributes. Such informal approaches may not be sufficient to reliably analyze certain quality attribute, e.g. performance, availability etc. A solution to this issue is to make and evaluate architectural design decisions by developing appropriate quality attribute models. In this respect, scenario-based SA evaluation methods developers and users can borrow a number of techniques from Prediction Enabled Component Technology (PECT) [23]. PECT is a systematic approach that applies quantitative reasoning framework and application specific information to predict the quality of component assembly.

SA design decisions are usually based on a quality attribute model (such as a scheduling model), which is an instance of a quality attribute reasoning framework. A reasoning framework provides the vocabulary and analytical machinery for describing and deducing particular system properties. Quality attribute reasoning framework consists of a set of independent and dependent parameters and their relationships [9]. The relationship between a reasoning framework, its independent and dependent variables and quality model, an instance of the reasoning framework, is shown in Figure 3.

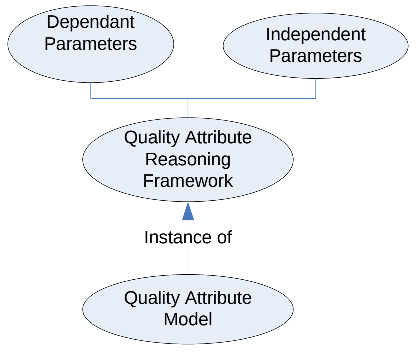

Figure 2: Relationship between quality attribute reasoning framework, its parameters and quality attribute model

We believe that the results of scenario-based architecture evaluation will be more reliable if an appropriate reasoning framework is used to analyze architectural design decisions with respect to the desired quality attributes. Quality attribute sensitive general scenarios can help the architect or evaluator select an appropriate reasoning framework. A general scenario contains quality attribute parameters that can identify one or more quality-attribute-reasoning frameworks [9]. If general scenarios for a SA evaluation have been structured according to the framework described in the previous section, such scenarios will have stimulus and response measures. The stimulus of a scenario can be treated as an independent parameter, while the response measures can be considered as dependent parameters. The theory of why manipulating independent parameter changes the value of the dependent parameter can explain the relationship in the quality model.

3. Skeleton of Relationships between patterns, tactics, evaluation and pattern validation

In this section, we present our assertion that software patterns are an important source of architectural related information that should be systematically captured and appropriately documented to improve the SA design and analysis processes. Moreover, we claim that the results of architecture evaluation may help rigorously validate the patterns used in that architecture. Figure 3 presents the relationships between patterns, architectural tactics, architecture evaluation and patterns validation.

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 [12]. Each pattern helps achieve one or more quality attribute in a system; however, each of them may also hinder other quality attributes. In pattern-oriented design, an architect develops a desirable SA by composing a number of architectural patterns and tactics. The format of a pattern's documentation requires the inclusion of problem, solution and quality consequences. Thus, each documented pattern has information on the description of the scenarios that characterize the quality attributes achieved by the pattern and associated quality consequences.

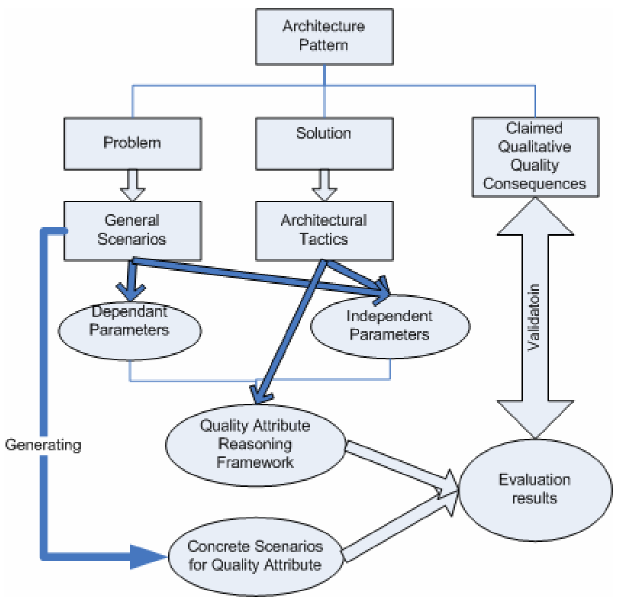

Figure 3: A skeleton of relationship between patterns, tactics, evaluation and pattern validation

These are the vital pieces of information required to perform SA evaluation. However, the existing format of pattern documentation does not make such information readily available to the architect and evaluators. This may be the reason that the SA evaluation does not usually use the information within the patterns. There is a need to provide complementary or alternative techniques to develop scenarios for SA evaluation and there is huge amount of information implicitly hidden in pattern descriptions, we contend that we can greatly improve the SA design and evaluation process by extracting and appropriately documenting the quality attribute specific information from the patterns.

The availability of general scenarios extracted from the patterns that promise to deliver the desired quality attributes during architecture design can help an architect to precisely articulate the quality requirements and ensure the quality requirements are complete [9]. From our point of view, the more important thing is the potential benefits of the extracted information to the architecture evaluation process.

Most of the scenario-based SA evaluation methods require the stakeholders to generate scenarios to evaluate the architecture. We argue that if the general scenarios that characterize the quality attributes satisfied by the patterns used in the SA are available to the stakeholders, it will improve SA evaluation and reduce the time and resources required to generate scenarios from scratch for each SA effort. Moreover, those general scenarios can also help the architect to select an appropriate reasoning framework to perform a more rigorous analysis of the architectural decisions with respect to the desired quality attributes. Additionally, the results of architecture evaluation can accumulate the evidence to either validate or reject a pattern's claim of providing certain quality attributes.

Let us clarify the concepts presented above with an example of a performance general scenario for EJB framework. Following is a performance sensitive general scenario extracted from a pattern named **Data Transfer Object** [24] using the framework described in Table 1.

*A large number of requests on an individual data entity attribute (**stimulus**) from a user interface (**source of the stimulus**) arrive at the system under normal condition (**environment**). The system (**stimulated artifact**) has to transfer the data (**response**) within a certain amount of time (**response measure**).*

The stimulus in this scenario suggests an event is arriving in some rate which may have some aspects of characteristics in terms of arrival distribution, number of event streams etc. All the aspects can be considered as independent parameters. The required measure of response is a hard-deadline response time, which is a dependent parameter. This information should help an architecture evaluator select an appropriate reasoning framework, which is a scheduling theory in this case. The chosen reasoning framework may provide some more independent parameters and can be calibrated into a quality model. During architecture evaluation, an architect can study how various architectural tactics can manipulate those independent parameters to achieve the desired quality attributes.

4. Extracting scenarios and tactics from patterns

In this section, we demonstrate the applicability of the theoretical concepts underpinning our research into mining software patterns for architecture evaluation and pattern validation. We present the architecturally significant scenarios and tactics extracted from well know software patterns. To improve readability and understandability, we document the extracted scenarios and tactics using structuring formats commonly used in SA discipline.

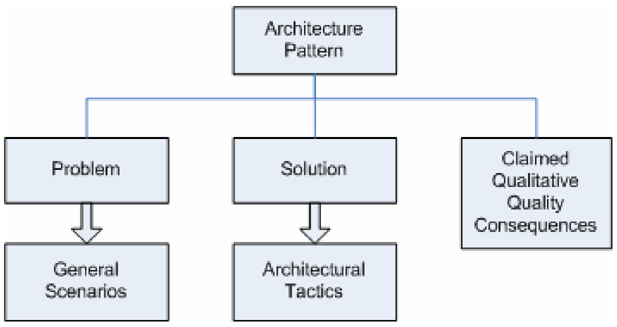

Figure 4: A diagrammatic view of a pattern

The process of distilling architecture sensitive information from pattern is implicit in the description because process issues are not within the scope of this paper. Figure 4 presents a diagrammatic view of the various parts that must be included in the pattern documentation and different types of the information that we can extract from a pattern. We use this diagram as a guide to the pattern mining process.

4.1 Distilling general scenarios from patterns

In this section, we show a few general scenarios extracted from known architectural patterns of Enterprise Java Bean (EJB) used in enterprise applications [24]. We have already stated and Figure 4 shows which part of a pattern contains what kind of information. Scenarios are described mostly in the problem element. While the quality attributes supported by the pattern are shown in the quality consequence part. Information on the relevant quality attributes is not usually presented in the early part of the pattern; especially quality attributes being negatively affected. We have extracted the quality attribute sensitive scenarios using a scenario development framework proposed in [9]. Stimulus, source of stimulus and environment can be found in the problem part of the investigated pattern. Response and stimulated artifact are commonly encountered in the solution part of the pattern. Explanations of the purpose of different parts within a pattern will reveal the stimulated artifact and expected response of the system. Response measures are usually pervasive, especially in the quality consequence part of a pattern's documentation.

We illustrate the concept by extracting a scenario from **Data Mapper pattern** [24]. The extracted scenario is:

A periodic data structure change request from stakeholders arrives when data use case changes. The system has to be modifiable according to the data structure change request within certain scope under certain time and maintenance cost.

Elements	**General Scenarios**
Source	Stakeholders
Stimulus	A periodic data structure change request
Environment	Normal condition
Artifact	System
Response	Modify
Response Measure	Certain time and maintenance cost

Table 2: Scenario 1

Similar general scenarios can also be extracted from Direct Access to Entity Bean, Data Transfer Object, Domain Data Transfer Object, Custom Data Transfer Object and Hash Factory [24]. However, not every extracted scenarios will focus on positive quality consequences. We can also extract scenarios by looking at negative quality consequences of a pattern and unexpected stimulus.

The other scenario has been extracted from the **Data Transfer Object** [24] pattern on data transfer performance:

A large number of data requests from an independent source arrive at the system under normal condition. The system has to transfer the data within a certain amount of time.

Elements	**General Scenarios**
Source	Independent source
Stimulus	A large number of data requests
Environment	Normal condition
Artifact	System
Response	Transfer the data
Response Measure	Within certain amount of time

Table 3: Scenario 2

Similar scenarios can be extracted from States Holder, Value Object, Detailed Object [24].

Both of the examples of scenario extraction from the architectural patterns are very high-level general scenarios. Patterns also contain rich context sensitive information, which can be used to refine the coarse-grained general scenarios into more fine-grained one. For example, by integrating the available contextual information, the performance general scenario can be refined into the following scenario:

A large number of requests on an individual data entity attribute from a user interface arrive at the system under normal condition. The system has to transfer the data within a certain amount of time without generating too many network calls

However, this scenario is still quite general. In order to make the general scenarios directly usable in SA evaluation, we need to convert them into concrete scenarios by providing system specific value for various elements like periodic, large, time and bandwidth etc.

Elements	**Refined General Scenarios**
Source	User interface
Stimulus	A large number of requests on data entity attribute
Environment	Normal condition
Artifact	System
Response	Transfer the data
Response Measure	Within certain amount of time

Table 4: Refined general scenario

By looking at the stimulus part, we find stimulus like data structure change request or requests on data entity attribute can indicate some characteristics of the independent parameters. However, this information alone may not be sufficient to enable a relatively inexperienced architect to analyse architectural decisions. Thus, we need to go to the next step of identifying and studying the tactics.

4.2 Distilling architectural tactics from patterns

We have stated that the general scenario extracted from a pattern provides valuable information that can help in selecting a reasoning framework. However, this information may not be sufficient to provide detailed selection criteria. Architectural tactics used in a pattern can also be very useful in the selection process. The solution and quality consequence parts of a pattern's documentation describe the tactics used in that pattern. Architectural tactics are documented in terms of manipulating some aspect of independent variables to achieve the target quality attribute. Thus, tactics used in a pattern provide more tangible information on the detailed aspects of parameters that can be manipulated to achieve the desired results.

Moreover, the solution and quality consequence parts also provide qualitative clues about the reasoning framework that is more suitable to the pattern. For example, a Factory intermediary tactic is introduced in the **DTOFactory** (Data Transfer Object Factory) pattern to isolate and localize the change. This tactic may be seen as an explanation for a modifiability impact analysis reasoning framework. The SA community has started documenting various tactics used in different patterns. Bass et. al. [9] documented various tactics in a hierarchical form. We have used that hierarchy to extract the tactics reported here.

Let us use an example from EJB usage pattern to illustrate the process of extracting general scenario and the tactics and selecting a reasoning framework. **DTOFactory** is frequently used in enterprise application to balance both performance and modifiability attributes. We have extracted two main general scenarios by looking into the problem, solution and consequence parts of the pattern.

General Scenario 1:
*A large amount of clients need to access server side object **(stimulus)**. The server has to serve the data within certain response time **(response measure)**.*

General Scenario 2:
*Frequent domain concept changes require changes to Data Transfer Object **(stimulus)**. The system has to be modified under certain time and maintenance cost **(response measure)**.*

The dependant variables we are interested in are modifiability in terms of effort and performance in terms of response time. Two stimuli here give valuable information to the independent parameters of the quality model but not in a very detailed fashion. However, we can get more information on the possible independent parameters by identifying tactics used in this pattern and the parameters being manipulated by those tactics. Following are the two tactics extracted from this pattern.

Tactic 1:

Independent Parameters	**Tactics**
Number of Requests	Manage demand by: • Managing event rate (Reduce request rate by bulk request)

Table 4: Tactic 1

This tactic is used in this pattern based on the assumption that network calls on the remote interface of an entity bean have severe performance burden on the server, which will affect the response time; and the number of calls generated on remote interface per unit of time can be reduced by putting a collection of data into a single Data Transfer Object. This is clearly one way of managing demand by managing event rate. However, packaging data into large transfer object will increase the amount of data transferred if requests are not aligned with the packaged data. A reasoning framework based on the number of requests and response time should be selected to evaluate the result using the concrete scenarios, which are the instances of the general scenario extracted from DTOFactory pattern. The result of this evaluation may also help validate the benefit of the DTOFactory pattern.

Tactic 2:

Independent Parameters	**Tactics**
Number of primary components affected	Localize expected modification by: • Limiting change options • Isolating expected changes • Maintaining semantic coherence

Table 5: Tactic 2

Table 5 shows the second tactic extracted from the pattern, which is other application of a "localize expected modification" tactic. When a change request on the Data Transfer Object happens, modification should be local with minimum number of ripple effects.

By introducing an intermediary, the change is limited in the factory method without affecting both the client and the entity bean (limit change options and isolate expected changes). In the meantime, semantic change on the client request should not change the server side semantic. By using the intermediary factory method, server side semantics coherence will be protected (maintain semantic coherence). It also provides the benefits of avoiding redeployment. The collection of tactics is used to manipulate the number of primary components affected. The general scenario has already indicated the dependent parameters, which affect maintenance. By associating the number of primary components affected and the effort required, we can calibrate the impact analysis model as our reasoning framework.

Having demonstrated how to extract the general scenarios and architectural tactics from software patterns, let us discuss how the extracted information can help in pattern validation. Deriving concrete scenarios from general scenarios extracted from the pattern assures us that we are evaluating the pattern with those scenarios, which are instances of the general scenarios the pattern is implementing. If certain concrete scenarios cannot be derived from the pattern's general scenarios that means the pattern may be not suitable for that problem. Moreover, the independent parameters that the pattern's tactics can manipulate must be part of the quality-attribute reasoning framework to ensure that the tactic (not any other factor) is achieving the benefit that pattern claims to provide. These two rules are utterly important in using the evaluation result as evidence for pattern validation.

5. Proof of concept - An illustrated example

In this section, we use an extended example to illustrate our approach to extract architecturally important information, general scenarios and tactics from software patterns to support architecture evaluation and to select and calibrate a reasoning framework and how the results from this analysis can be used to accumulate the evidence to validate the pattern.

The process of selecting and calibrating reasoning framework may vary from one quality attribute to the other depending on the context. For example, the process of choosing and adjusting a reasoning framework for modifiability impact analysis can be different to the one used for performance quality attribute. In the following section, we use the architectural information extracted from a pattern to select and calibrate a performance model in Enterprise Java Beans (EJB) context using the process and example presented in [25] by Liu et. al.

5.1 Overview of the process

Liu et. al. [25]has proposed a process to select and calibrate performance reasoning framework using architectural and application specific information under the PECT framework. Their process first selects a performance model based on an individual's experience and then calibrates the chosen model according to two sources of information: 1) Architecture specific information. This comes from the environment and implementation constraints. For example, in order to calibrate a performance model for EJB architecture, Contain Managed Persistence (CMP), and a find-by-non-primary-keys entity bean needs to be integrated into the model. Architectural tactics like Read-Only, caching and session façade also provide some information. 2) Application specific information. This consists of typically scenarios from the application domain, e.g. typical online stock trading scenarios have a large amount of read-only requests.

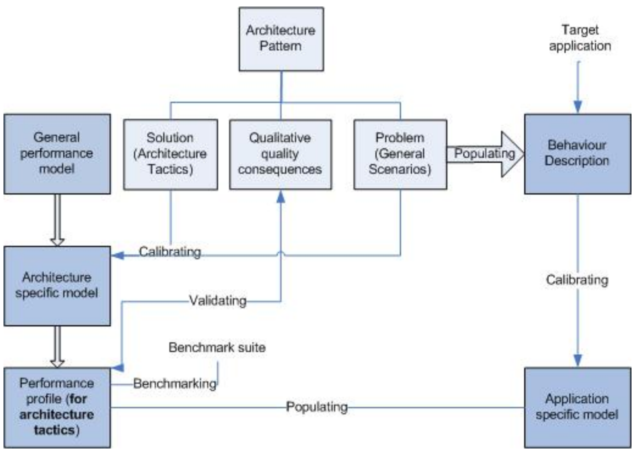
Architecture Pattern
Target application
General performance model
Solution (Architecture Tactics)
Qualitative quality consequences
Problem (General Scenarios)
Populating
Behaviour Description
Calibrating
Architecture specific model
Validating
Calibrating
Benchmark suite
Performance profile (for architecture tactics)
Benchmarking
Populating
Application specific model

Figure 5

Having studied the process model of Liu et. al, we assert that our research into mining software patterns to extract scenarios and tactics can complement their process. Figure 5 shows how the architectural information extracted from patterns using our approach can be used in the calibrating process of Liu et. al. Our claim is based on the assumption that general scenarios and architectural tactics can provide the information required to calibrate the general model for architecture-specific model. General scenarios can be instantiated to generate application specific concrete scenarios and calibrate the application-specific model. The performance profile derived from the pattern can be bundled with the pattern as valuable information on pattern use and architecture evaluation of the pattern-based application.

By using this approach, we believe architecture evaluators can use the qualitative reasoning information found in patterns to help calibrate quantitative attribute models. Even in non-quantitative situation, the information can help adjust the general qualitative model. We can also use this approach to compare different patterns, that satisfy the same or similar general scenarios.

5.2 Pattern Description

The pattern investigated in this example is a RM (Read-Mostly) pattern of EJB application. When there are a large number of read-only requests, an entity bean on the server side can be semantically divided into a read-only bean and a write-only bean. By using caching on the read-only bean, if a read request hits the cache, a data source access is avoided. We can achieve better performance by using the RM pattern. We can use this pattern description along with the extraction techniques and extracted information structuring techniques described earlier to demonstrate our approach to mining software patterns to improve the SA evaluation process and accumulate evidence to validate the RM pattern.

Following is the description of the four steps along with the examples of our approach.

Step 1 – Extract scenarios from RM pattern

By using the six element framework for structuring scenarios, we have extracted the following general scenario from The RM pattern used in the example application. The constituent parts of the scenario are shown in table 4. Here is the description of the scenario: *A large number of client **(stimulus source)** requests, mostly read-only requests **(stimulus)** arrive at the server at normal condition **(environment)**, the server (artifact) needs to process the requests **(response)** within certain response time **(response measure)**.*

We can use the stimulus (client request) and the response measure (hard deadline response) parts of the above-mentioned scenario and some experience of client server environment to identify the need of some sort of queuing model. This can model the server's behavior in processing a request from a client as the reasoning framework. However, only this information may not be enough to select and calibrate the most suitable model. To gather more information, we move onto the next step of extracting tactics from the pattern.

Step 2 – Extract tactics from RM pattern

We can identify two tactics in the RM pattern: managing demands by includes introducing caching and reducing event rate. The caching is used for read-only requests. If a corresponding entity bean instance is in the cache, the cached item will be returned so the event rate on serialization and synchronization with entity bean can be reduced. These events usually have significant impact on performance. In an EJB application, the time involved in a server replying client request has two parts: EJB container response time and data source response time. Container response time includes T_1 (service time to synchronize with the database) and T_2 (service time to access the entity bean). Data source response time includes T_{find} (service time to find the data), T_{load} (service time to load the data) and T_{store} (service time to store the data).

By using caching and reducing the event rate, the number of requests that need resource intensive activities can be decreased. This will shorten the total response time for a large number of read-only requests. The identified tactics can manipulate the following parameters:

- Within the EJB container, reduced event rate results in reduced need for synchronization with database and the access entity bean. The parameters, which can be manipulated, include number of requests (R1) related to T1 (service time to synchronize with database) and number of requests (R2) related to T2 (service time to access the entity bean).
- Within the data source, reduced rate can reduce the service time to load and store the data. Since, no matter whether the request is read-only or not, we need time to find the item, the time to find the data is not affected by the event rate. The parameters which can be manipulated in this case, are number of requests (R3) related to Tload (service time to load the data) and number of requests (R4) related to Tstore (service time to store the data).

Step 3 - Select and calibrate reasoning framework

Having gathered the information based on the above-mentioned scenarios and tactics, an architect will be more confident in calibrating a QNM (queuing network model) as an appropriate quality-attribute reasoning framework. Taking the constraints imposed by other design and technological issues, e.g. EJB-specific information, CMP (Container Managed Persistent) issue, into account, Liu et. al. [26] has shown how they adjusted the QNM reasoning framework to the response time quality model for container and data source. Here is their logic to calibrate QNM to response time model:

$$\int_{container} = R_1 T_1 + R_2 T_2$$
$$\int_{datasource} = R_3 T_{load} + R_4 T_{store} + T_{find}$$

All the number of requests here can be expressed in terms of ratio of read-only requests (p) and cache hit rate (h_r):

$$\begin{aligned}
R_1 &= (p + np_c)h_r + (1 - p_c - p)h \\
R_2 &= (p + np_c)(1 - h_r) + (1 - p_c - p)(1 - h) \\
R_3 &= 1 - p_c - ph_r + np_c(1 - h_r) \\
R_4 &= 1 - p_c - p
\end{aligned}$$

R_1, R_2, R_3, R_4 are independent parameters as mentioned above. The introduction of h_r and p in the equation and their clear manipulation on the independent parameters has reflected the tactics on managing demands in this pattern. During architecture evaluation, we can test ratio of read-only requests and hit ratio regarding their relationship with the response time to verify whether this architecture satisfies the requirements and the tactics contribute to achieve the desired response time which is the benefit claimed by the pattern used in the architecture.

Step 4 – Accumulate proof to validate the pattern

Having extracted scenarios and tactics from RM pattern and selected and calibrated a suitable quality-attribute model, we can use this information to evaluate a SA that is using RM pattern. The evaluation results can provide some evidence to validate the effectiveness of the pattern. A real world application may provide much stronger evidence. Since a pattern is an abstract concept, any concrete implementation of the pattern if evaluated should qualify as evidence. Liu et. al.[26], designed and built a benchmark application based on Stock-Online [27], which uses the RM pattern. We will use the application as a form of validation of the RM pattern. Following is a concrete scenario that is an

instance of the general scenario extracted from the RM pattern (recall the two prerequisites of the validation process mentioned in section 1):
A large number of stock transaction related requests (get/set/create accout/stockholders/stockitems/stockTx, 36/43 of them is read-only request arrives at the server at normal condition), the server needs to process the request within certain response time.

The response time is unspecified in the scenario because we are only investigating the concrete scenario and we are interested in a wide range of the response times. Thus, it is obvious that the concrete scenario is the instance of the general scenario satisfied by the pattern. Let us make sure that the second requirement is also met: tactics must manipulate only those parameters which belong to the reasoning framework's independent parameters. This is shown in the RM pattern response time quality model used in [26] as the tactics and corresponding parameters are clearly modeled in the reasoning framework for the response time.

We can take approaches to use the result of the architectural evaluation as an evidence to validate the pattern:

- Compare the pattern with a design without using the tactics on managing demands.
- Manipulate the cache hit ratio to see the results of response time.

Both approaches are valid. Liu et. al.[26] used the first approach in order to compare two designs; the second happens to be the one without using the tactics. They used the process of evaluating architecture for performance quality attribute. The result of their evaluation shows that the contribution of tactics in the pattern is very significant. Considering the fact that we are evaluating against concrete scenarios derived from general scenario extracted from the pattern, the result can be considered as the validation of the pattern.

In this extended example, we have dissected a performance modeling work done within the PECT framework according to demonstrate the practical application of our approach to distill general scenarios and architectural tactics from a pattern. The extracted information can help select and calibrate a general performance reasoning framework – the QNM model. Then, the extracted general scenario can be used to develop concrete scenario for the architecture to be evaluated. These two steps facilitate the architecture evaluation process which requires concrete scenarios and reasoning framework. Since the parameters being manipulated by the tactic are the independent parameters of the reasoning framework, we can claim that the architecture evaluation results validate the benefits of using RM pattern as asserted by the pattern documentation.

6. Future work and Conclusions

6.1 Future work

This paper reports our initial work on improving SA evaluation activities for pattern-oriented systems by extracting and documenting the architecturally critical information from well known software patterns. We are developing a rigorous and systematic approach to identify and capture any piece of information, i.e. scenarios, tactics, etc., that can be useful during the architecture design or evaluation. The work reported in this paper cannot be described as complete in its current form as there are a number of things that need to be done before the users, i.e. architects/designers/SA evaluators, of our approach can experience significant benefits in terms of reduced architecture evaluation time, improved confidence in using patterns to compose architecture for desired quality attributes, ease of comparing patterns and so on. In the short term, we plan following tasks to refine and validate our approach:

- To validate the usefulness and generality of our approach by applying it to various applications and quality attributes models.
- To develop a quantifiable means of showing benefits of using it.
- To develop a reusable repository of the general scenarios and tactics for well-known patterns.
- To expand our approach into evidence-based empirical software engineering by providing a systematic way of validating design patterns.

6.2 Conclusions

In this paper we present an approach to mine software patterns for extracting architecturally significant information to support SA evaluation. We assert that software patterns are a vital source of architectural information, which can be systematically extracted and appropriately documented to support and improve SA evaluation process. We base our claim on our investigation into the relationship between architectural patterns, software quality attributes, and scenario-based SA evaluation approaches. We argue that general scenarios and architectural tactics extracted from software patterns can be used as a useful input to architecture evaluation and reasoning framework selection and calibration processes. Moreover, we believe that the architecture evaluation results may provide an evidence of pattern validation.

Our main conclusion is that the existing usage of patterns in software engineering does not utilize the large amount of architectural information implicitly embedded in them. Patterns are mainly used to compose architectures for complex and large systems. One of the reasons for using patterns is the widely accepted belief that they help satisfy various quality attributes requirements. An architect or SA evaluator needs to come up with the scenarios that can precisely characterize the required quality attributes and then try to identify those patterns which promise to satisfy those scenarios. We contend that this is a quite time consuming and expensive process. We assert that if patterns are documented along with the scenarios they satisfy and the tactics which contribute to the benefits provided by a pattern, the architecture design and evaluation process can greatly be improved.

Given our use of general scenarios and architectural tactics along with the independent variables that can be manipulated by the tactics in architecture design and analysis process, we also conclude that the information extracted from the patterns can help an architect select and calibrate the most suitable quality-attribute reasoning framework. We use only those concrete scenarios that are instances of the general scenarios extracted from the pattern. Moreover, the parameters manipulated by the pattern's tactics are the independent parameters of the reasoning framework of the quality attributes to be satisfied by the pattern. We further conclude that the results of the evaluation of that pattern may be used to rigorously validate the pattern.

Our major research object is to improve the process of SA design and evaluation by gathering and documenting architectural information in a format that is readily useful in making design decisions with an informed knowledge of the consequences of those decisions. Especially we want to reduce the time, resources and skill level required to effectively and efficiently evaluate a system's SA with respect to various quality attribute models. We believe that mining software patterns to distill and document SA sensitive information is one of the most important steps towards that goal.

7. References

[1] D. L. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules," *Communication of the ACM*, vol. 15, pp. 1053-1058, 1972.

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

[3] N. Lassing, D. Rijsenbrij, and H. v. Vliet, "The goal of software architecture analysis: Confidence building or risk assessment," First BeNeLux conference on software architecture, 1999.

[4] R. Kazman, M. Barbacci, M. Klein, and S. J. Carriere, "Experience with Performing Architecture Tradoff Analysis," 21th International Conference on Software Engineering, New York, USA, 1999.

[5] R. Kazman, L. Bass, G. Abowd, and M. Webb, "SAAM: A Method for Analyzing the Properties of Software Architectures," Proceedings of the 16th International Conference on Software Engineering, 1994.

[6] N. Lassing, P. Bengtsson, J. Bosch, and H. V. Vliet, "Experience with ALMA: Architecture-Level Modifiability Analysis," *Journal of Systems and Software*, vol. 61, pp. 47-57, 2002.

[7] M. Ali-Babar, L. Zhu, and R. Jeffery, "A Framework for Classifying and Comparing Software Architecture Evaluation Methods," Australian Software Engineering Conference, Melbourne, 2004.

[8] P. Bengtsson and J. Bosch, "An Experiment on Creating Scenario Profiles for Software Change," University of Karlskrona/Ronneby, Sweden HK-R-RES-99/6-SE, 1999.

[9] L. Bass, F. Bachmann, and M. Klein, "Deriving Architectural Tactics-A Step toward Methodical Architectural Design," Software Engineering Institute, Carnegie Mellon University CMU/SEI-2003-TR-004, 2003.

[10] L. Zhu, M. Ali Babar, and R. Jeffery, "Distilling Scenarios from Patterns for Software Architecture Evaluation," First European Workshop on Software Architecture, 2004.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns-Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.

[12] F. Buschmann, *Pattern-oriented software architecture : a system of patterns*. Chichester ; New York: Wiley, 1996.

[13] L. Prechelt, B. Unger, W. F. Tichy, P. Brossler, and L. G. Votta, "A controlled experiment in maintenance: comparing design patterns to simpler solutions," *Software Engineering, IEEE Transactions on*, vol. 27, pp. 1134-1144, 2001.

[14] K. Beck, J. O. Coplien, R. Crocker, L. Dominick, G. Meszaros, F. Paulishch, and J. Vlissides, "Industrial Experience with Design Patterns," 18th International Conference on Software Engineering, Berlin, Germany, 1996.

[15] *IEEE Standar 1061-1992, Standard for Software Quality Metrics Methodology*. New York: Institute of Electrical and Electronic Engineers, 1992.

[16] J. A. McCall, "Quality Factors," in *Encyclopedia of Software Engineering*, vol. 2, J. J. Marciniak, Ed. New York, U.S.A.: John Wiley, 1994, pp. 958-971.

[17] ISO/IEC, *Information technology - Software product quality: Quality model*, ISO/IEC FDIS 9126-1:2000(E).

[18] G. Abowd, L. Bass, P. Clements, R. Kazman, L. Northrop, and A. Zaremski, "Recommanded Best Industrial Practice for Software Architecture Evaluation," Software Engineering Institute, Carnegie Mellon University CMU/SEI-96-TR-025, 1997 1997.

[19] R. Kazman, G. Abowd, L. Bass, and P. Clements, "Scenario-Based Analysis of Software Architecture," *IEEE Software*, Nov. 1996.

[20] N. Lassing, D. Rijsenbrij, and H. v. Vliet, "How Well can we Predict Changes at Architecture Design Time?," *Journal of Systems and Software*, vol. 65, pp. 141-153, 2003.

[21] N. Lassing, D. Rijsenbrij, and H. v. Vliet, "On Software Architecture Analysis of Flexibility, Complexity of Changes: Size isn't Everything," 2nd Nordic Software Architecture Workshop, 1999.

[22] R. Kazman, S. J. Carriere, and S. G. Woods, "Toward a Discipline of Scenario-based Architectural Engineering," *Annals of Software Engineering, Kluwer Academic Publishers*, vol. 9, 2000.

[23] K. Wallnau, "Volume III: A Technology for Predictable Assembly from Certifiable Components," Software Engineering Institute, Carnegie Mellon University CMU/SEI-2003-TR-009, 2003.

[24] F. Marinescu, *EJB design patterns : advanced patterns, processes, and idioms*. New York: John Wiley, 2002.

[25] Y. Liu, A. Fekete, and I. Gorton, "Predicting the performance of middleware-based applications at the design level," Fourth International Workshop on Software and Performance (WOSP 2004), Redwood City, California, USA, 2004.

[26] Y. Liu, A. Fekete, and I. Gorton, "Design Level Performance Modeling of Component-based Applications," School of Information Technologies, University of Sydney TR543, 2003.

[27] I. Gorton, *Enterprise Transaction Processing Systems: Putting the CORBA OTS, Encina++ and OrbixOTM to Work*: Addison-Wesley, 2000.