An Ontology-Driven Software Architecture Evaluation Method

Aida Erfanian
Electrical and Computer Engineering Faculty
Shahid Beheshti University
Evin, Tehran, Iran
a.erfanian@mail.sbu.ac.ir

Fereidoun Shams Aliee
Electrical and Computer Engineering Faculty
Shahid Beheshti University
Evin, Tehran, Iran
f_shams@sbu.ac.ir

ABSTRACT

Software architecture evaluation has a crucial role in the life cycle of software intensive systems. In this paper we propose an approach to empower a software architecture evaluation method called the Architecture tradeoff Analysis Method (ATAM). Our approach is highly focused on the effective reusability of software architecture knowledge. We propose two ontologies focusing on the role of Attribute-Based Architectural styles (ABAS) in software architecture development and analysis. We show the effectiveness of our approach by presenting three case studies.

Categories and Subject Descriptors

D.2.4 [**Software/Program Verification**]: [Model checking, Validation]; D.2.1 [**Software Architectures**]: [Patterns (e.g., client/server, pipeline, blackboard)]

General Terms

Documentation, Verification, Design

Keywords

Software Architecture, ATAM, Ontology, Architectural Knowledge, Quality Attribute

1. INTRODUCTION

Software architecture has a prominent role in the life cycle of software intensive systems [4]. So, it is important to effectively analyze and evaluate software architecture to extract the incomplete specifications from the point of the stakeholders' views. From this point of view, it is obvious that it is actually the documentation of software architecture which transforms the evaluation process to an effective one. On the other hand, current documentation approaches like UML mostly focus on the notational specifications [6], while they have weaknesses in documenting the architectural decisions [9] that lead the architecture evaluation [7]. Architectural decisions take different quality attributes in to account and are the key factors in exposing risks and sensitivity points of the architecture while the evaluation is in progress [4]. Furthermore, architectural styles play an important role during evaluation [16], [15] to extract and derive the possible combination of design decisions from earlier experience [11].

There are several scenario based software architecture evaluation methods like Software Architecture Analysis Method (SAAM) [7], Architecture Tradeoff Analysis Method (ATAM) [7] that try to extract software architectural decisions in order to identify risks and evaluate the possibility of the combination of multiple decisions for the goal of the quality attributes fulfillment [7]. It is believed that appropriate information can be extracted and documented systematically to support the software architecture evaluation process [16]. Hence, it is important to have a well-formed and expressive infrastructure to document the current architectural knowledge to reuse the past evaluation experience in a structured way.

Zhu, Babar, and Jeffery have tried to support the evaluation process by the means of patterns mining in order to extract general scenarios for the quality attributes of a pattern [16]. The goal of their work is to make the pattern based general scenarios available to save the time of the evaluation. Any way they did not present a solution of the evaluation team to appropriately extract the possible combination of patterns and their quality attributes.

Akerman and Tyree have worked on architectural decisions and exploited them to build an ontology to support the development of software architecture [14], [3]. They claimed that their ontology is useful for validating a software architecture. Although, their work is useful to trace different parts of an architecture and validates the relationships among them, they do not present a reasoning framework to assure the correctness of relationships. In addition their work cannot support the creation of an effective repository that has much more advantage over the traditional repositories, because they define the elements of the ontology via descriptive sentences. Moreover, to enter the validation phase the ontology needs to be instantiated. The fact is that the most of the today's architectural documentations are not formal and not ontological yet. So, the evaluation team cannot restrict others to give the ontological sets of documentations as the inputs of evaluations. To drive the architecture evaluation process we need a supporting ontology that lead the evaluation process even when we don't have an ontological specification of an architecture.

In this paper we focus on building infrastructures or meta-models for analysis and evaluation of software architecture taking the ATAM in to consideration. We propose two meta-models by exploiting ontology to address the documentation and architectural knowledge reusability during the evaluation by building a reasoning framework based on ABAS. Hence, our proposed ontologies improve traceability and provide an effective way to save and reuse the architectural knowledge for the present and future evaluations.

We believe that our proposed ontologies can be applied even in the cases that there is not a complete set of architectural documentations both in early and late stages of the development. With this ontology of the meta-model we can bring the process of development and evaluation of software architecture together to increase the effective impacts that each of them can have on another. This process is based on ATAM [7] and we call it the *Ontology-Driven ATAM*.

Section 2 is a glance at existing ATAM that is the base of our work. Then we present the proposed ontologies in section 3. We define our approach and prove the effectiveness of it in section 4. The last section is the conclusion of our work.

2. THE ATAM

In this section we overview a scenario-based evaluation method that is the base of our work called Architecture Tradeoff Analysis Method (ATAM) [7]. This method is developed by the Software Engineering Institute (SEI) and currently is in use for the purpose of the evaluation of the fulfillment of multiple quality attributes in software architecture.

ATAM can be useful both in *early* and *late* phases of the development of software architecture. Early evaluation takes place in any stage of software architecture development process to ensure that the already made architectural decisions will fulfill the desired quality attributes [7]. Late evaluation takes place even when the system has been implemented; this is the case when the organization uses some sort of legacy systems [7]. ATAM is consisted of nine steps that are divided into four groups: presentation (Present the ATAM, present business drivers, present architecture), investigation and analysis (Identify architectural approaches, generate quality attribute utility tree, analyze architectural approaches), Testing (Brainstorm and prioritize scenarios), and Reporting (present results). These steps form a process flow. Figure 1 shows the process flow of ATAM.

As Figure 1 shows the business drivers and the software architecture are the first inputs of ATAM. From these inputs the quality attributes and architectural approaches are elicited. Then the scenarios and architectural decisions are exposed in order to feed the analysis process. The outputs of the analysis is the identification of tradeoffs, sensitivity points, non-risks, and risks which are distilled into risk themes. All of these steps may be occurred iteratively to reach to the desired software architecture.

The Software Engineering Institute (SEI) has encountered the followings as the proven benefits of applying the ATAM:

- Clarified quality attribute requirements.
- Improved architecture documentation.
- Documented basis for architectural decisions.
- Identified risks early in the life-cycle.
- Increased communication among stakeholders.

The outputs of ATAM can be summarized as:

- The architectural styles identified.
- A *utility tree* - a hierarchic model of the driving architectural requirements.
- The set of scenarios generated and the subset that were mapped onto the architecture.
- A set of quality-attribute specific questions that were applied to the architecture and the responses to these questions.
- A set of identified risks.
- A set of identified non-risks.

In the next section we will introduce our ontologies to support an ATAM-based evaluation.

3. OUR PROPOSED ONTOLOGIES

Stakeholders and their concerns, architectural decisions, architectural styles, and the architectural assets, play important roles in the ATAM because these are the issues which show that whether the architecture will fulfill the highest priority quality attributes or not. So, we have focused on them and their relationships in building the ontological meta-models.

To improve the traceability of such ontology we have to carefully build the semantics of the relationships between the entities. The main classes of this ontology are the followings:

- Stakeholder.
- Concern.
- Architectural Decision.
- Scenario.
- Architectural Style.
- Architectural Asset.
- Quality Attribute.

By the means of our proposed ontologies we have tried to codify the analysis steps of ATAM. This work is done by considering semantic relationships among architectural styles, architectural assets and architectural decisions which play important roles during the analysis steps of ATAM. An architectural style consists of component types and their topology, specifications of data patterns and interactions among the components including an informal specification of the advantages and disadvantages of the style [9].

In the first ontology that we call it Software Architecture Ontology (SAO) each architectural style implements several architectural decisions which are concerned by different quality attributes [4]. Architectural decisions can be classified into groups for more clarification [14], [12]. The justification of the architectural decision can be defined by the rational of the architectural decision [12]. Each architectural decision can have other alternatives. An architectural decision can be in different states (approved, pending, rejected). Each architectural decision leads to some constraints. Figure 2 depicts the semantic relationships. The ontology shown on Figure 2 contains other semantic

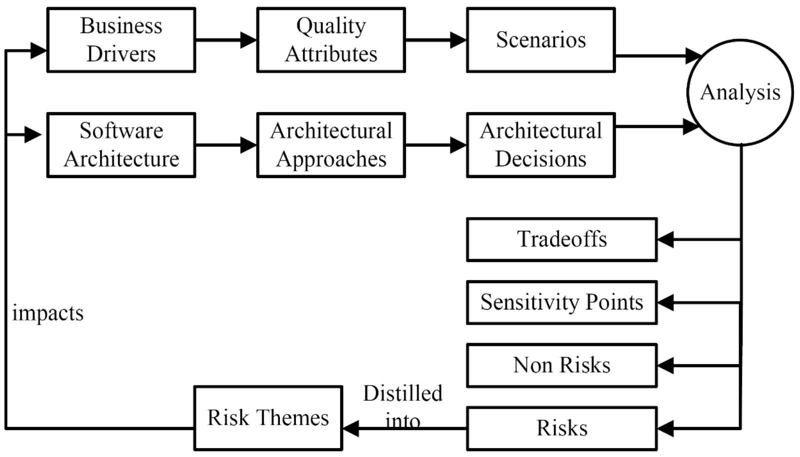

Figure 1: The process Flow of ATAM [2].

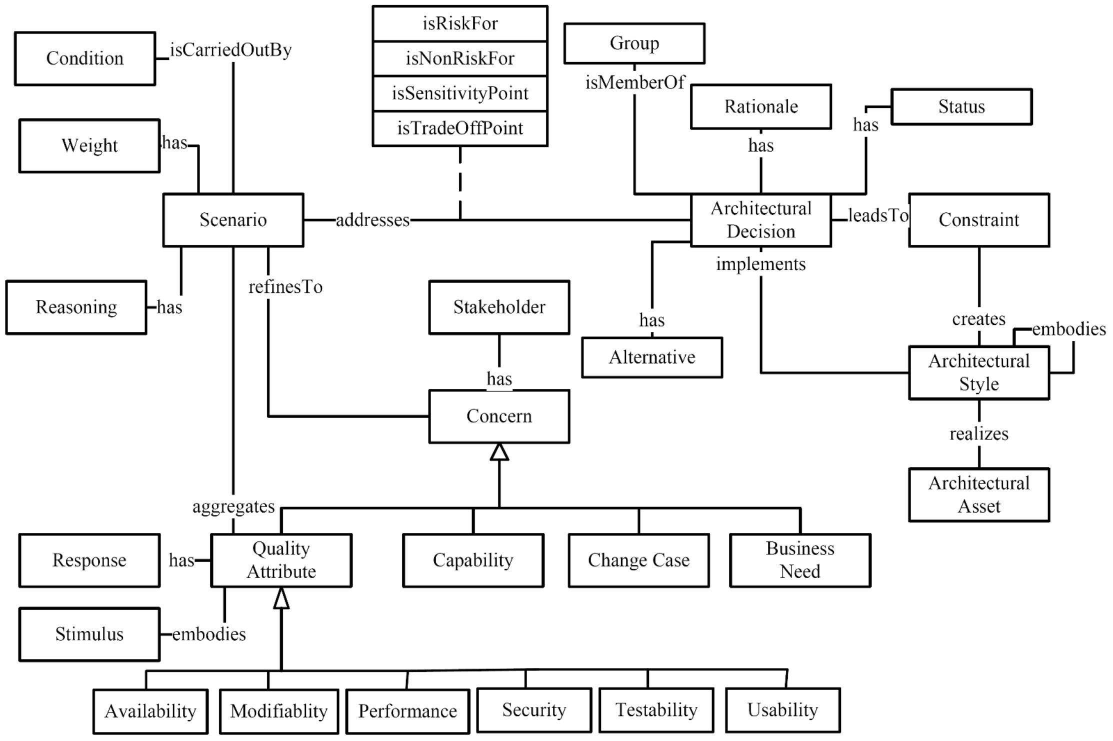

Figure 2: The ontology of the proposed approach.

elements too. These elements include Scenario that is addressed by one or more Architectural Decision(s). Each Scenario is carried out by zero or more condition. While the evaluation is in progress, a scenario may gain a Weight. Each scenario has Reasoning. A scenario aggregates one or more Quality Attributes. Quality Attribute is one of the Concerns of a Stakeholder. Another concern of a Stakeholder is Capability (The functional requirement of the system). Business need (Business goal) and Change Case are the subclasses of a Stakeholder's Concern.

In addition to the described ontology which is instantiated with respect to the software architecture that is under the evaluation, we have created another ontology to support the tradeoff decisions in several steps of an ATAM. We built this ontology based on Attribute-Based Architectural Styles (ABAS) [5]. We call the second ontology the ontology-driven ABAS or OABAS. Figure 3 depicts OABAS. As Figure 3 shows, each architectural style has some element types. Each element type has some semantic constraints and interaction mechanisms. For example, the pipe and filter style [4] has element types of pipe and filter. Filters do not control either upstream or downstream elements (Semantic constraints), and data are transported through pipes (Interaction mechanism).

To illustrate relationships among architectural styles and tactics and quality attributes, the Active Object design pattern [13] is a good example. The active object design pattern implements the information hiding tactic that has a good effect on modifiability.

OABAS is fully instantiated by the architectural styles and their tactics and the corresponding quality attributes. The knowledge of OABAS can evolve and refine as the evaluation team gains more experience. The main purpose of OABAS is Architectural knowledge reusability. It is worth to know that the quality attribute subclasses are extensible and can consist other types of quality attributes like flexibility.

We implemented the proposed ontologies (SAO and OABAS) through an open source ontology tool called the protégé [10]. To do the ontology oriented evaluation the following rules must be obeyed during adding a new individual to the SAO:

Rule1. The SAO must import the OABAS. We have implemented SAO and OABAS by a semantic web language called OWL-DL. So, each of our proposed ontologies has a unique URI. Therefore, each of them has a namespace. Each ontology can import any other ontology just like the source codes written by traditional programming languages do.

Rule2. Any individual of the architectural decision class of SAO must be selected from the available set of OABAS's tactics.

Rule3. Any individual of the architectural style class of SAO must be selected from the available set of OABAS's architectural style class.

Rule4. Any individuals of the quality attribute class of SAO must be selected from the available set of OABAS's quality attributes.

Rule5. The semantic relationship between the architectural style and architectural decision in SAO will be inferred automatically while selecting an OABAS's tactic's individual as a member of architectural decision through a consistency rule that we have declared in SWRL-Jess tab [1] of protégé:

ConsistencyRule1.

$OABAS : Tactic(?t) \wedge$
$OABAS : ArchitecturalStyle(?as) \wedge$
$OABAS : implements(?as, ?t) \wedge$
$ArchitecturalDecision(?t) \rightarrow$
$ArchitecturalStyle(?as) \wedge implements(?as, ?t)$

Rule6. There is an additional class in SAO to infer the inconsistent individuals during the evaluation. We call this set *InconsistentSet*. The quality attributes that the architectural styles and decisions have conflict with, become the members of this set by following ConsistencyRule2.

ConsistencyRule2.

$OABAS : Tactic(?t) \wedge$
$OABAS : ArchitecturalStyle(?as) \wedge$
$OABAS : implements(?as, ?t) \wedge$
$OABAS : QualityAttribute(?qa) \wedge$
$OABAS : hasBadEffectOn(?t, ?qa) \wedge$
$QualityAttribute(?qa) \wedge$
$ArchitecturalStyle(?as) \wedge$
$ArchitecturalDecison(?t) \wedge$
$implements(?as, ?t) \rightarrow InconsistentSet(?qa)$

In the next section we describe our Ontology Driven ATAM through illustrative case studies.

4. ONTOLOGY-DRIVEN ATAM

In the first phase of ATAM the evaluation team often does the go/no-go decision [7]. We have tried to codify criteria for this step using our proposed ontologies. To enter the go/no-go decision step, the evaluation team uses the candidate system documents that are generated through the step two. In step two the evaluation team asks the architecture team about the coarse-grained architectural styles and architectural decisions and the relevant quality attributes.

The first case study shows the situation in which the SAO and OABAS would help the go/no-go decision. However, there are another situations and steps of ATAM which the described ontologies would act as effective means. The second and third case studies clarify such situations.

4.1 The First Case Study

In this section we describe a case study in which the evaluation team uses ATAM to reveal the capability of a system that is in the maintenance phase, in the fulfillment of some quality attributes. This system is operating in an environment that its conditions are changing. Considering the changes to the environment the project manager of the system would like to know whether their system can operate in such environment or not.

In the first phase of ATAM after the evaluation leader described ATAM in step 1, during step 2, the architect of the system tells the evaluation team that the architecture is mainly built based on the layered architectural style and each layer embodies the pipe-and-filter style and the main purpose of using these styles is modifiability that was firstly

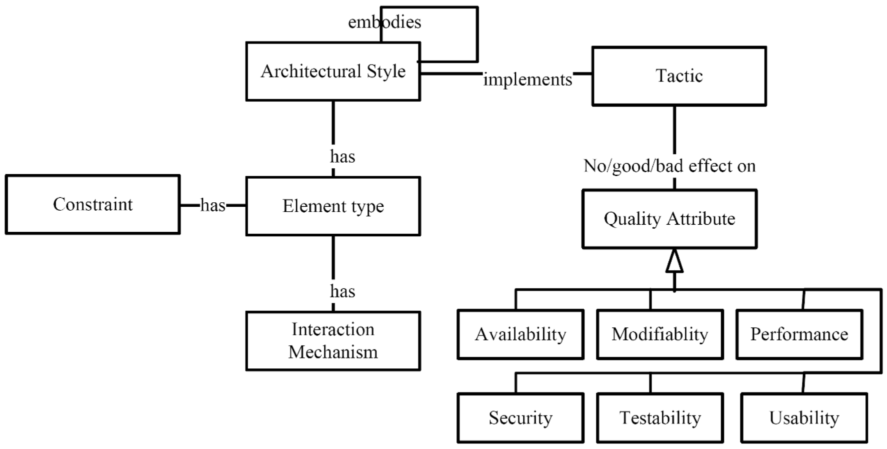

Figure 3: Ontology driven ABAS (OABAS).

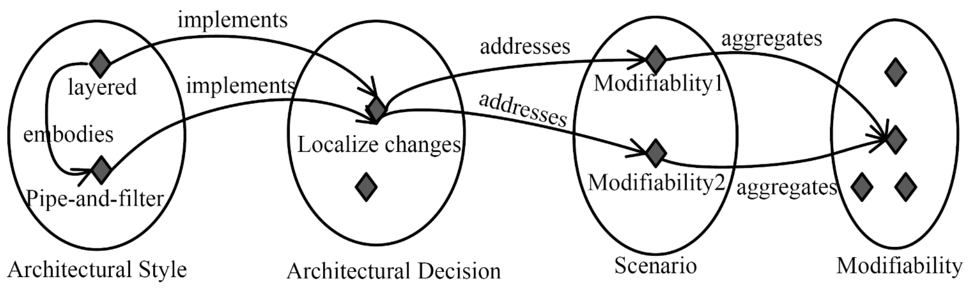

Figure 4: An example of SAO ontology

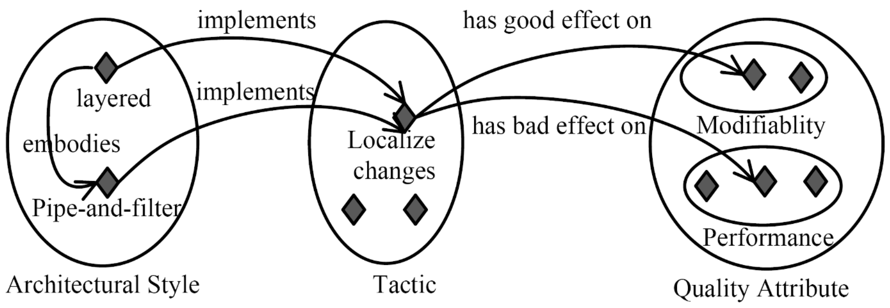

Figure 5: An example OABAS ontology

the main concern of the project manager. In step 2, ontology of the architecture instantiates based on SAO. Figure 4 shows a sample part of the ontology. As Figure 4 shows, in this architecture, *layered* architectural style embodies *pipe-and-filter* style (in each layer). Both of the styles implement *localize changes* as an architectural decision to address the one of the modifiability individuals. There is a class for each quality attribute type that can have multiple individuals for a quality attribute since each architectural decision has its own pros and cons on quality attributes. For example, *Localize changes* is a good decision for modifiability, and may have bad influences on *Reduce computational overhead* that is a good decision for one of the individuals of performance, but it may have no special effect on scheduling policy that is a good decision for another individual of performance. Thus, from these descriptions one can deduce in mind that s/he may use the combination of *Localize changes* and *scheduling policy* together to increase the modifiability and performance of the system in his/her special view. So, the semantic coherence between the architectural decisions will be increases.

The next step is the go/no-go decision. The OABAS, which is an ontology contains the architectural styles and their advantages and disadvantages leads this step. To lead the evaluation process effectively, OABAS needs to be instantiated by taking the ABAS and the previous experiences in to account. By considering the meta-model of OABAS that is mentioned in the previous section, Figure 5 shows a sample part of OABAS. Figure 5 shows that *layered* and *pipe-and-filter* styles implement *localize changes*. *Localize changes* has a good effect on *modifiability* but it has a bad effect on *performance*.

In step 2, the project manager has told the evaluation team that s/he would like to know whether their system is capable to perform in a highly interactive environment or not (high performance). In step 3 (go/no-go decision), the evaluation team executes ConsistencyRule2. Since, the situation of this case study satisfies ConsistencyRule2, the *Performance individual* becomes a member of *InconsistentSet*. Having looked at the Inconsistent set the evaluation team realizes that the current architecture is not a good candidate for the high performance quality attribute. The high traceability nature of OABAS also enables the evaluation team to easily query for the possible tactics of performance that *Localize changes* does not affect on. If there is no possible way to mitigate the situation, then the go/no-go decision can be made with more consciousness.

In situations that a software architecture is more complex, and combination of decisions are made for it, OABAS is a helpful means that effectively saves the time and energy. That is because complex architectures usually exploit different architectural styles and tactics to fulfill the desired quality attributes. OABAS contains the possible relationships among architectural styles, tactics, quality attributes and the effects each can impose on another. These predefined effects (based on best-practices and experiences of the evaluation team) that are consist of good/bad/no effect enable us to study the possible combinations of these elements in an architecture just by tracing or querying OABAS.

4.2 The Second Case Study

Using OABAS, the architecture can be built from the scratch, evolving and refining gradually while it is iteratively under the evaluation (early evaluation). For example, consider a situation in which the architecture team utilizes SAO and OABAS ontologies to build software architecture from the scratch. The architecture team has received the high run-time modifiability, and performance as the most important quality attributes of a web based system that is going to be built.

The software architecture team can simply query OABAS for the tactics that have good effects on modifiability. A list of tactics that have good effect on modifiability appears as the result of query. Selecting the most matching tactic with the run-time modifiability and the problem domain that would be *differ binding time*, the architect can trace it to reach to its relative architectural style. In this case the relative architectural style would be *Interpreter*. The *differ binding time* tactic has a relative individual in the Modifiability class. Thus, the architect team can query OABAS for the possible tactics of performance that does not affect *differ binding time* related modifiability individual, in a bad manner. In this case, the result would be *Increase computational efficiency* and *Scheduling policy*. The result architectural decisions (tactics) from this step are the candidate architectural decisions to enter the next step of analysis as inputs. Figure 6, illustrates the part of OABAS that is relevant to this case study.

4.3 The Third Case Study

The forth step of ATAM follows the similar activities of the first and second case study. In this step, the mapping between the architectural decisions and the architectural assets will be instantiated in SAO considering the architectural documentations and asking the architect (The work on architectural assets is not the purpose of this paper).

The next (fifth) step of ATAM is the stage where the quality attribute utility tree is going to be generated. This step can be started using the candidate architectural decisions (possibly the architectural decisions resulted from the second case study) as inputs. During this step the evaluation team and the architect team identify the architectural decisions from the candidate set and prioritize and refine them in the mentioned SAO ontology.

The step 6, in which the architectural approaches will be analyzed, starts with the instantiated SAO that embodies the utility tree as a part of its semantics. It is worth to know that using an ontology tool like protégé, executing some SQL-like queries and extracting the utility tree during the analysis is very simple.

The analysis results (scenarios and their weights and mapping of them to the architectural decisions, architectural styles and architectural assets, stakeholders and their concerns) will be inserted in SAO following the third section's rules. During these last steps identification of analysis question, tradeoff points, risks and non-risks, can be simplified by means of using the previous SAOs of experience and the existing OABAS. For example, consider a situation in which the same relationship between a quality attribute scenario and its relative architectural decision exists in the previous experiences ontologies that was known as a risk (By the highly expressive, well formed and traceable nature of SAO, extracting such knowledge is simple). Thus, the potential risks can be exposed automatically when the analysis of the architectural approaches. Therefore, the current architectural decision will be identified as a potential risk, too.

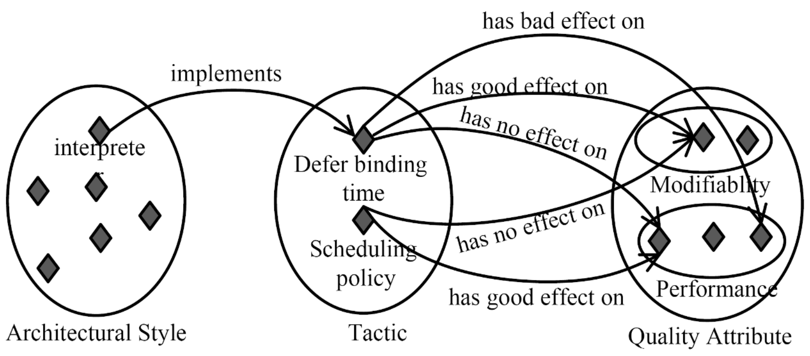

Figure 6: A sample part of OABAS for the second case study.

The potential risks can act as efficient inputs of the analysis steps.

5. CONCLUSIONS

Software architecture evaluation is an important activity that ensures whether an architecture fulfills the concerns of stakeholders in early stages of the life cycle software-intensive systems, or not. An evaluation will clarify the weaknesses of an architecture and the candidate solutions in a more expressive and effective manner if it utilizes the best practices and past architectural knowledge in a simple well formed and expressive way.

An evaluation method will be much more effective if the architectural knowledge of the evaluations is reused by the software architecture team during the architecture development process with low costs.

Using ontologies we express the vocabularies and semantics of our problem domain with an expressive, explicit and well-defined manner [8]. In this paper, we proposed OABAS and SAO to address the evaluation phase knowledge reusability while generating the ATAM necessary knowledge to drive the ATAM steps in an ontological manner. OABAS is used to empower the evaluation decisions criteria through funding a semantic basis for modified ABAS. OABAS contains the good/bad/no effect semantics between the tactics and different classified quality attributes. Each quality attribute type can have multiple relationships with different tactics. In this way, the architecture team or the evaluation team can expose the possible combination of tactics for a set of quality attributes. SAO is used to document the architecture while the evaluation is in progress, considering OABAS. Performing ATAM by the support of OABAS and SAO the ontology driven ATAM forms as the case studies show. Moreover, we have focused on the role of *architectural styles* in development and evaluation of software architecture. Choosing an architectural style is one of the main steps of Attribute-Driven Design [4], and can act as a pivot point to drive an evaluation method like the ATAM to exploit the architectural approaches more effectively. In addition, knowledge sharing is an advantage of ontologies which can be mentioned as another benefit of our approach.

6. REFERENCES

[1] Protege swrljesstab, September 2007. http://protege.cim3.net/cgi-ben/wiki.pl?SWRLJessTab.

[2] The architecture tradeoff analysis method (atam), 2008. http://www.sei.cmu.edu/architecture/ata_method.html.

[3] Akerman, A., and Tyree, J. Using ontology to support development of software architectures. *IBM Syst. J. 45*, 4 (2006), 813–825.

[4] Bass, L., Clements, P., and Kazman, R. *Software Architecture in Practice, Second Edition.* Addison-Wesley Professional, April 2003.

[5] Bass, L., and John, B. E. Achieving usability through software architectural styles. In *CHI '00: CHI '00 extended abstracts on Human factors in computing systems* (New York, NY, USA, 2000), ACM, pp. 171–172.

[6] Clements, P., Garlan, D., Little, R., Nord, R., and Stafford, J. Documenting software architectures: views and beyond. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering* (Washington, DC, USA, 2003), IEEE Computer Society, pp. 740–741.

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

[8] Daconta, M. C., Smith, K. T., and Obrst, L. J. *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management.* John Wiley & Sons, Inc., New York, NY, USA, 2003.

[9] de Boer, R. C., Farenhorst, R., Clerc, V., van der Ven, J. S., Deckers, R., Lago, P., and van Vliet, H. Structuring Software Architecture Project Memories. In *8th International Workshop on Learning Software Organizations (LSO)* (Rio de Janeiro, Brazil, 2006), pp. 39–47.

[10] Gennari, J. H., Musen, M. A., Fergerson, R. W., Grosso, W. E., Crubézy, M., Eriksson, H., Noy, N. F., and Tu, S. W. The evolution of protg&#233;: an environment for knowledge-based systems development. *Int. J. Hum.-Comput. Stud. 58*, 1 (2003), 89–123.

[11] Jansen, A., and Bosch, J. Software architecture as a set of architectural design decisions. In *WICSA '05: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 109–120.

[12] Kruchten, P. An ontology of architectural design

decisions in software intensive systems. In *2nd Groningen Workshop on Software Variability* (Dec 2004), pp. 54–61.

[13] Schmidt, D. C., Rohnert, H., Stal, M., and Schultz, D. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. John Wiley & Sons, Inc., New York, NY, USA, 2000.

[14] Tyree, J., and Akerman, A. Architecture decisions: Demystifying architecture. *IEEE Softw. 22*, 2 (2005), 19–27.

[15] Zhu, L., Babar, M. A., and Jeffery, D. R. Distilling scenarios from patterns for software architecture evaluation - a position paper. In *EWSA* (2004), pp. 225–229.

[16] Zhu, L., Babar, M. A., and Jeffery, D. R. Mining patterns to support software architecture evaluation. In *WICSA '04: Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA'04)* (Washington, DC, USA, 2004), IEEE Computer Society, p. 25.