Tactic Based Approach to Building Domain Architecture

1Sung Nam Lee, 2Keokyoon Ko, 3*Suntae Kim, 2Sooyong Park

1Defense Acquisition Program Administration, Seoul, South Korea
dapalee@korea.kr
2Dept. of Computer Science & Engineering, Sogang University, Seoul, South Korea
{maniara, sypark}@sogang.ac.kr
3*Dept. of Computer Engineering, Kangwon National University,
Samcheok Si, South Korea,
stkim@kangwon.ac.kr

1. Introduction

In building application architecture, a software architect consults domain architecture, which is composed of domain concepts and architectural approaches to achieving common quality requirements of the application domain[1]. Based on the domain architecture, a software architect focuses on diverse strategies to handle application specific functional requirements(FRs) and non-functional requirements(NFRs). However, due to the nature of elements of the domain architecture that can be realized in diverse manners, it is difficult to capture and represent elements of domain architecture.

There have been some approaches to addressing the issues. Almeida et al.[3] proposed a domain architecture development process including activities, guidelines and roles. They introduced how a system can be decomposed into sub-modules and when architecture patterns can be applied. However, all activities introduced in the paper should be carried out manually without any automatic support. Kim et al. [5] suggested an approach to specifying domain concepts at UML meta-level and generating UML models from domain patterns. Our approach is inspired by this approach and extends it to handle domain non-functional requirements in more systematic manner.

This paper proposes an approach to building domain architecture using domain requirements and architectural tactics [1] by utilizing our previous study [6][7]. Our previous study proposed a method to build semi-automatic software architecture using architectural tactics represented in RBML [4], which is a UML based pattern specification language at UML meta-model level. In this paper, application requirements are classified into domain FRs/NFRs and application FRs/NFRs. Then, domain component model is derived from domain FRs, and then architectural tactics are selected by domain NFRs in a quantitative manner [7]. Finally, domain architecture is derived from composing the domain component model and architectural tactics according to composition rules. In this paper, we use Job candidate management application to illustrate our approach.
2. Building Domain Architecture

Our approach for building domain architecture consists of four steps as below:

– Step 1: Requirements Classification

All application requirements are classified into four types: Domain FR, Domain NFR, Application specific FR and Application specific NFR. While domain FR and NFR indicate functional and non-functional requirements that can be commonly discovered in the application domain, application specific FR and NFR represent FR and NFR that can be searched only in the application. Thus, two sets are mutually exclusive each other. Henceforth, application FR and NFR indicate application specific FR and NFR. The following shows domain requirements of Job candidate application management domain.

• Domain FR: An applicant shall write a resume through our system.
• Domain NFR: When an applicant writes a resume, the system shall provide the service with over 99% availability. If the service is not available, the system shall detect and notify it to the system administrator.

– Step 2: Developing Domain Component Model

![Use Case Model](image1)

![Domain Component Model](image2)

*Figure 1. Identifying Domain Component Model from Use Case*

Domain component model presents components covering FRs in the application domain. Identifying domain components starts with use cases that describes domain FR. Use case can be realized by system components that are in charge of interacting with actors and orchestrating control flows, and business components that cover a business entities in the domain[2]. Either <<include>> or <<extend>> relationship is transferred into an association between system components. Fig. 1 shows an example of identifying domain components from ‘Write a result’ and ‘Login’ use cases, which are from DFR of the step 1.
Tactic Based Approach to Building Domain Architecture

– Step 3: Selecting Architectural Tactic Model

Based on the quantitative approach, architectural tactics are selected to satisfy domain NFRs. Our previous work introduced costs of architectural tactics in Tactic Use Case Point (TUCP) and equations to compute Selection Factors (SF) [7]. To select an appropriate architectural tactic for a domain NFR, a software architect estimates Architectural Tactic Contribution Factor (ATCF) of each candidate architectural tactic for the NFR with a number ranging from 0.0 to 1.0. Then the approach automatically calculates sensitivity value to increase confidence for the estimation. The sensitivity value is understood as the lowest value to guarantee the selection. Architectural tactics for the domain NFR1 for fault detection can be candidate tactics, which are Exception, Ping/Echo, Heartbeat and composition of Ping/Echo and Heartbeat. A software architect estimates how much each candidate tactic can be contributed to satisfy the given NFR with ATCF. After computing the SF, Ping/Echo has the highest SF (0.66) so that it is selected as the most appropriate tactic for the domain NFR. Its sensitivity value is 0.75, which means that the any ATCF value of the Ping/Echo over at least 0.75 guarantees the selection of the Ping/Echo tactic.

– Step 4: Composing Domain Component Models and Tactic Models

Our study on architectural tactic semantics and their synergetic relationship provides architectural tactic semantics described in RBML [6]. Then, the tactic semantics is composed with domain component model in this step. This is because NFR generally denotes a quality constraint of the components for satisfying FR. In the case of DFR and DNFR, |ResumeSysComp should play a role of a component that sends the echo message back to the |PingSender component. Thus, |ResumeSysComp and |PingReceiver should be bound together. The process can be described in the binding rule as below:

B1. Ping/Echo::|PingReceiver! DomainComponentModel::|ResultSysComp

Fig. 2. Domain Architecture for DFR1 and DNFR1

Fig. 2 shows the domain architecture for DFR1 and DNFR1. In the figure, |FaultMonitor, |PingSender and |PingReceiver are from Ping/Echo tactic semantic, and remainders are from domain component model. The architecture is produced by the composition rule B1 for composing |ResultSysComp and |PingReceiver. As all
component in the domain architecture are described in RBML, it constraints model level elements and their relationship. Thus, a component that plays a role of ResumeSysComp also should play a role of PingReceiver.

3 Conclusion and Future Work

In this paper, we have presented an approach to building domain architecture from the set of application requirements and architectural tactics. In addition, we have proposed a method to identify domain component model from application requirements specified in use case model and binding rules for composing components from domain component model and those from architectural tactic semantics. Based on our approach, domain architecture can be established in more structured way. Also, the architecture can capture diverse model level components as it adopted RBML to describe components. As future work, we have a plan to apply it into Job candidate management application domain more comprehensively and develop tool support to facilitate the model composition process.

Acknowledgements.

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology. (2012M3C4A7033348)

* Corresponding Author: Suntae Kim, Ph.D. - Dept. of Computer Engineering, Kang-won National University, Kangwon Do, South Korea

References