ISSN 2234-4438
융합정보논문지 제7권 제2호 pp. 67-76, 2017 DOI : https://doi.org/10.22156/CS4SMB.2017.7.2.067

오픈 소스 소프트웨어 재사용을 위한 소프트웨어 아키텍처 설계

최용석, 홍장의*
충북대학교 컴퓨터과학과

Designing Software Architecture for Reusing Open Source Software

Yongseok Choi, Jang-Eui Hong*
Department of Computer Science, Chungbuk National University

요 약 소프트웨어 활용의 수명주기가 단축되고, 다양한 형태의 사용자 기능을 지원하기 위하여 소프트웨어 아키텍처 개발의 중요성이 부각되고 있다. 소프트웨어 아키텍처가 새로운 기능을 갖도록 확장에 유연하고 안정적으로 개발된다면, 새로은 시장의 요구에 빠르게 대응할 수 있다. 본 연구에서는 새로운 기능 개발의 수명주기가 짧아지고 있고, 안정적인 소프트웨어 시스템 개발을 위하여 오픈 소스의 설계 복구를 통한 재사용을 고려하는 아키텍처 설계 기법을 제안한다. 소프트웨어 아키텍처를 기반으로 소프트웨어 시스템을 개발하기 위하여 오픈 소스를 사용하는 경우, 매우 신속한 개발이 가능해 질 뿐만 아니라, 이미 검증된 오픈 소스를 사용함으로써 개발 시스템에 대한 신뢰성도 증진시킬 수 있다.

키워드 : 소프트웨어 아키텍처, 오픈 소스, 설계 복구, 컴포넌트 재사용, 속성 기반 설계

Abstract Along with shortening the life cycle of software utilization and supporting various types of user functions, the importance of software architecture development has been emphasized recently. If a software architecture is developed flexibly and reliably for expansion to support new functionality, it can quickly cope with new market demands. This paper proposes an architecture design method based on design recovery of open source software to reuse the software in the development of sustainable software system. When using open source software to develop a software system based on software architecture, we can develop a software system rapidly and also can improve the reliability of the system because we use the already proven open source software in the development.

Key Words : Software architecture, Open source, Component reuse, Design recovery, Attribute-driven design

1. 서론

소프트웨어의 수요가 급증하고, 모든 영역에서 소프트웨어의 사용이 증가됨에 따라 보다 신속하고 안정적인 소프트웨어 개발이 요구되고 있다. 오픈 소스는 이러한 신속한 소프트웨어 시스템 개발과 안정적인 소프트웨어 개발을 지원한다[1]는 측면에서 최근 많은 소프트웨어 개발자에게 관심의 대상이 되고 있다. 그런데, 오픈 소스를 체계적으로 사용하기 위해서는 개발하고자 하는 소프트웨어 시스템에 적합한 소스인가를 판단해야 한다. 작은 단위 규모의 소프트웨어 개발에서는 큰 무리없이 오픈 소스를 쉽게 선택하고 사용할 수 있지만, 대규모 소프

Received 2017-03-10 Revised 2017-04-10 Accepted 2017-04-20 Published 2017-04-30
*Corresponding author : Jang-Eui Hong (jehong@chungbuk.ac.kr)

트웨어를 개발하는 경우에는 오픈 소스에 대한 구체적인 정보, 즉 인터페이스 컨트랙(Contracts)이나 오픈 소스의 내부 로직과 구조가 반드시 고려되어야 한다[2,3].

그러나 일반적으로 오픈 소스가 갖는 설계 정보는 대규모 소프트웨어 시스템을 개발하기에 부족한 경우가 많다[4]. 현재 공개되는 전체 오픈 소스 중에서 해당 컴포넌트의 설계정보를 제공하는 경우는 전체의 30%를 넘지 않고 있다고 알려져 있다[5,6].

따라서 본 연구에서는 먼저 오픈 소스에 대한 분석을 통해 설계 정보를 복구하는 방법을 제안하였다. 이는 오픈 소스를 이용하여 소프트웨어를 개발한다는 프로젝트의 방침을 염두에 두고, 대규모 시스템의 소프트웨어 아키텍처를 개발하기 위함이다. 다시 말해서 소프트웨어 아키텍처를 설계하기 위한 아키텍처 드라이버(Architectural Drivers)로써, 오픈 소스의 재사용을 고려한다는 것이다.

기존에 소스 코드로부터 설계정보를 추출하는 연구가 많이 진행되어 왔다. E. Constantinou의 연구는 소스 코드를 정적분석하여 DAG(Directed Acyclic Graph)를 생성하고, 이를 이용하여 아키텍처 정보를 추출하기 위한 방법을 제안하였으며, L. Heinermann의 연구에서는 20개의 자바 오픈소스 프로젝트 분석을 통해, 블랙 박스(Black-Box) 재사용과 화이트 박스(White-Box) 재사용의 분포를 비교하여, 블랙박스 재상용의 유용성을 확인하였다[7,8]. 또한 Doxygen 도구나 Bunch 도구와 같은 역공학 지원도구들을 이용하여 설계 정보를 도출할 수 있다[9-11]. 그러나 이러한 연구들은 단지 설계 정보에 대한 도출 방법을 제안하는 것이며, 이를 어떻게 설계과정 특히 소프트웨어 시스템의 아키텍처 개발에 사용할 것인지에 대해서는 제시하지 못하고 있다.

본 논문의 구성은 다음과 같다. 2장에서는 소스코드로부터 설계 정보를 복구하는 기존의 관련 연구들을 살펴보고, 3장에서는 본 연구에서 제시하는 오픈 소스의 재사용을 고려한 아키텍처 설계 기법, SADOSS (Software Architecture Design based on Open Source Software)의 절차를 설명한다. 4장에서는 SADOSS 절차 중에서 오픈 소스로부터 설계 정보를 추출하기 위한 기법을 제안하고, 5장에서는 이러한 설계정보를 이용한 소프트웨어 아키텍처 개발 기법을 설명한다. 6장에서는 사례 연구를 통한 제안 기법의 적용성을 평가하고, 마지막으로 결론 및 향후연구를 기술한다.

2. 관련 연구

소스 코드에 대한 설계정보 추출을 위한 기존의 연구는 설계 복구(Design Recovery)라는 관점에서 많은 연구가 진행되었다. 이들 중에서 최근에 제안하는 대표적인 연구들을 살펴보면 다음과 같다.

먼저 E. Constantinou의 연구는 소스 코드로부터 DAG를 생성하고, 아키텍처를 구성하는 4 계층, 즉 User Interface Layer, Controller Layer, Business Logic layer, 그리고 Infrastructure layer에서의 DAG 그래프를 구성하고, 이들을 조합한다[7]. 이 때, 각 DAG의 노드는 클래스와 연결되어 있다. L. Heinermann의 연구는 소스 코드의 부분적인 수정을 통해 이루어지는 화이트 박스 재사용과 코드에 대한 수정 없이 이루어지는 블랙 박스 재사용이 기존 자바 프로젝트에서 얼마나 빈번히 발생하는지를 분석하였다[8]. 이 결과 20개 프로젝트 중에서 9개의 프로젝트가 화이트 박스 재사용을 시도하였으며, 전체 코드 사이즈의 10%미만이 재사용 되었다. 블랙 박스 재사용의 경우, Java API 기반의 재사용이 많은 부분을 차지하고 있으며, 평균적으로 45%의 프로젝트별 재사용율을 보였다.

그리고 Y. Cai의 연구는 보다 정밀한 설계 정보를 얻기 위하여 시스템을 모듈로 분리하는 방법을 제시하였는데, 이 과정에서 모듈간에 존재하는 광역 변수, 추상화된 인터페이스와 같은 분할의 제약사항을 해결하기 위한 규칙을 제안하였다[12]. 이를 통해 소스 코드의 재사용율을 향상시키고자 하였다.

이외에도 UML 모델 정보를 이용한 코드의 자동 생성, 안드로이드 어플리케이션의 재사용을 위한 설계 정보 복구 기법, 그리고 자바 오픈 소스 재사용을 위한 설계 패턴 추출과 같은 연구들이 있다[13-15]. 그러나 앞서 조사 분석된 연구들이 비록 설계 정보를 추출하여 소프트웨어 시스템 개발에 활용하고자 하였으나, 아키텍처의 융통성과 확장성을 고려하는 소프트웨어 아키텍처의 개발 보다는 단지 존재하는 소스 코드의 재사용을 향상시키기 위한 목적으로 수행 되었다.

3. SADOSS 기법의 정의

3.1 SADOSS 절차

본 연구에서 제안하는 SADOSS 기법은 Fig. 1에서 보

는 것과 같이 크게 두가지 단계, 즉 코드 리버싱(Code Reversing)과 아키텍처 설계(Architecture Design)로 진행된다. 코드 리버싱 단계는 오픈 소스로부터 설계정보를 추출하여 저장 관리하는 단계이고, 아키텍처 설계 단계는 요구사항을 기반으로 개발 대상 소프트웨어의 아키텍처를 개발하는 단계이다. 이때 코드 리버싱 단계의 결과물을 활용한다.

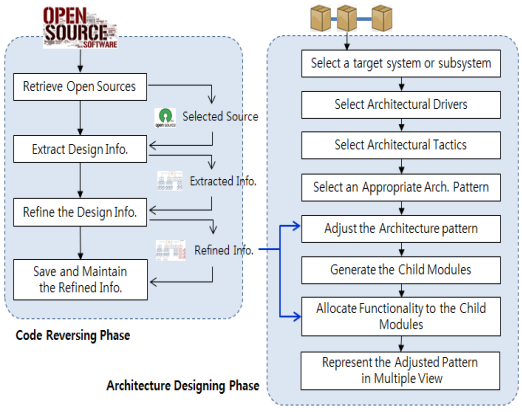

Fig. 1. The Procedure of SADOSS

3.2 코드 리버싱 단계

Fig. 1에 제시한 코드 리버싱 단계에서 수행하는 세부 스텝별 활동은 다음과 같다.

(CR-1) 요구사항 분석 결과를 이용하여 오픈 소스를 탐색한다.

(CR-2) 선택한 코드로부터 도구를 이용하여 설계 정보를 자동 추출한다.

(CR-3) 도구로부터 자동으로 획득한 설계 정보에 대한 가시성 증대와 재사용성 향상을 위한 설계 정보의 정제 및 부족한 정보의 보완 작업을 수행한다.

(CR-4) 정제된 설계 정보를 저장 관리한다.

3.3 아키텍처 설계 단계

Fig. 1의 아키텍처 설계 단계에서는 다음과 같은 작업의 수행을 통해 대상 시스템에 대한 아키텍처를 개발한다. 이러한 아키텍처 개발 절차는 Len Bass가 제시된 ADD(Attribute-Driven Design) 기법을 근간으로 확장한 것이다[16].

(AD-1) 아키텍처를 개발할 대상 시스템 또는 서브 시스템을 선정한다.

(AD-2) 대상 시스템에 대한 아키텍처 드라이버를 선택한다. 이러한 드라이버의 하나로 “요픈 소스의 재사용”을 선정한다.

(AD-3) 아키텍처 전술(Tactics)을 선정한다. 아키텍처 전술은 개발 대상시스템에서 제공해야하는 품질 속성을 만족하기 위한 다양한 방법 중에서 적절한 방법을 선택하는 활동이다[17].

(AD-4) 전술을 만족하는 아키텍처 패턴을 선택한다. 아키텍처 패턴은 개발 대상 시스템이 어떠한 요구사항을 갖는가에 따라 매우 다양하게 분류되고 제시되어 있다[18,19].

(AD-5) 선택한 패턴을 아키텍처 드라이버를 기반으로 조정한다. 이때 본 연구에서는 “코드 리버싱” 단계에서 획득한 설계 정보를 반영한 조정 작업을 수행한다.

(AD-6) 조정된 아키텍처 패턴의 인스턴스를 생성하여 어플리케이션의 아키텍처를 생성한다. 이때 설계 정보를 고려한 인스턴스를 생성한다.

(AD-7) 아키텍처를 구성하는 컴포넌트에 세부 기능을 할당한다. 이러한 기능 할당도 “코드 리버싱” 단계의 결과를 적용하는 컴포넌트를 우선적으로 할당한다.

(AD-8) 사용자 시나리오를 고려하여 다양한 관점의 아키텍처를 표현한다.

4. 설계정보 추출 방법

앞의 3.2절에서 SADOSS 기법의 “코드 리버싱” 단계의 수행 활동을 정의하였다. 이러한 활동 중에서 (2)와 (3)단계가 어떻게 수행되는 지 구체적으로 설명한다.

4.1 설계정보 자동추출

선택된 오픈 소스로부터 설계 정보를 얻기 위해 잘 알려진 Doxygen 도구를 사용한다. Doxygen은 주석이 달린 코드를 읽어내서 문서를 만들어 내는 소프트웨어 레퍼런스 문서 생성기로서 보통 HTML파일이나 설정에 따라 Work 파일로 결과를 추출할 수 있다.

Doxygen은 선택한 소스 파일에 대하여 Class Hierarchy, Function Call 등, 보고자 하는 설계 정보를 설정한 후 프

로그램을 실행시키면 Fig. 2와 같은 결과를 얻을 수 있다.

이와 같은 정보는 소프트웨어 설계 과정에서 사용하기 위하여 최소한의 정보로 문서화되어야 한다. 추출된 설계 정보를 문서화하기 위하여 본 논문에서는 GoF에서 사용하는 설계 패턴의 정의 형식을 사용하며, 이의 내용은 Table 1과 같다[19].

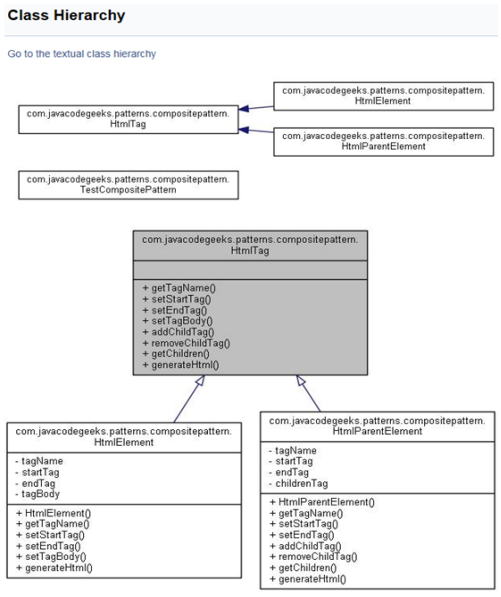

Fig. 2. An Example of Design Information of Source Code using Doxygen.

4.2 설계 정보의 정제 및 보완

자동화 도구인 Doxygen을 이용해서 얻는 설계 정보는 Table 1에서 정의하는 정보를 충분히 제공하지 못하는 문제가 있다. 물론 Bunch와 같은 다른 도구들에 의해서는 보다 더 많은 정보를 확보할 수 있지만, Table 1의 내용을 전체적으로 추출하기는 어렵다.

따라서 소프트웨어 엔지니어는 자동화 도구에 의해 채워지지 않는 Table 1의 해당 정보를 소스 코드의 분석이나 해당 오픈 소스에 첨부된 매뉴얼 또는 개발자가 제공하는 정보 분석을 통해 추출해야 한다.

본 논문에서는 오픈 소스로부터 추출된 정보의 활용을 위해 Table 1에서 제시하는 12개의 항목 중에서 아키텍처 설계 과정에 꼭 필요하지 않은 정보, 즉 (1) 패턴의 이름과 분류, (2) 의도, (3) 패턴의 다른 이름, (8) 패턴의 적용 영향, (11) 예제, (12) 관련 패턴에 대해서는 필요하다면, 별도의 추출 과정 없이 진행하도록 정의하였다.

Table 1. Design Information Specification of GOF

No	SECTION	DESIGN INFORMATION
1	Pattern Name and Classification	• Unique and descriptive name to identify and explain a pattern
2	Intent	• Reason why this pattern is used
3	Known As	• Another name (alias name)
4	Motivation	• Situation or scenarios to use this pattern
5	Applicability	• Contexts of this pattern to be used.
6	Structure	• Graphical representation of the pattern. (may use class diagram and collaboration diagram)
7	Participants	• List of classes within the pattern and their meaning of the classes
8	Consequences	• pros and cons for using the pattern
9	Implementation	• Syntax and semantics of the pattern and explanation of how to implement
10	Sample Code	• Example code using programming language
11	Known Uses	• Examples applied to practical systems
12	Related Patterns	• other patterns related to this pattern • differences between this pattern and others

5. 아키텍처 설계 방법

앞의 3.3절에서 정의한 SADOSS 기법의 “아키텍처 설계” 단계에 대하여 구체적으로 설명한다. 설명을 위한 간단한 예제 시스템으로 IoT 홈 오토메이션에서 차고의 도어 제어 시스템(Garage Door Open Control System, GDOCS)을 사용한다.

5.1 아키텍처 드라이버 선정

아키텍처 드라이버는 시스템의 목표를 달성하기 위해 제시된 요구사항 중에서 시스템 품질, 시스템 전체에 대한 제약사항, 또는 시스템 구성요소와 상호 작용에 대한 것이다. GDOCS의 아키텍처 드라이버는 다음과 같이 선정하였다.

- 서로 다른 홈 오토메이션 플랫폼과 연동할 수 있어야 한다.
- 장애물이 탐지되면 0.1초 이내에 도어를 다시 올려야 한다.
- 오픈 소스를 이용한 시스템 기능을 개발한다.

5.2 아키텍처 전술 선정

위와 같은 드라이버를 기반으로 아키텍처 개발을 위한 전술을 정의할 수 있다. 예제의 장애물 탐지 동작에 대한 전술은 Fig. 3과 같다.

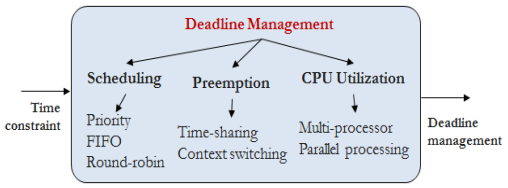

Fig. 3. Architectural Tactics for Timing Constraints

Fig. 3에 나타난 전술 중에서 본 논문에서는 Deadline을 보장할 수 있는 스케쥴링(Scheduling) 방법을 채택하기로 한다.

5.3 아키텍처 패턴 선정

선정된 아키텍처 드라이버와 아키텍처 전술을 만족시키는 아키텍처 패턴을 선정한다. GDOCS는 IoT 환경에서 작동하는 자동 제어 기능과 홈 오토메이션 시스템과의 통신을 필요로 하는 시스템이다. 따라서 Fig. 4와 같은 패턴을 선택할 수 있다.

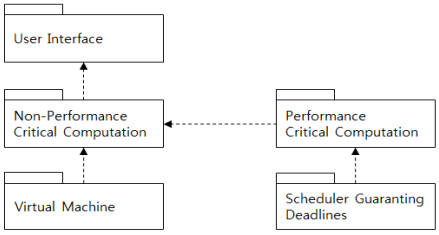

Fig. 4. Selected Architecture Pattern for GDOCS

5.4 아키텍처 패턴 조정

Fig. 4에 주어진 패턴을 기반으로 오픈 소스를 사용한 기능 개발에 대한 아키텍처 드라이버를 적용하면 Fig. 5의 조정된 패턴을 얻을 수 있다.

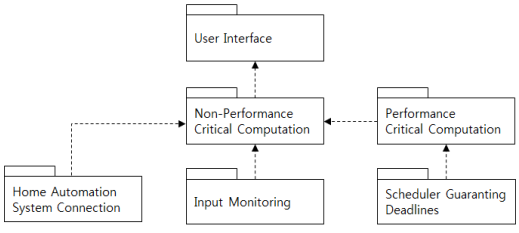

Fig. 5. Adjusted Architecture Pattern for GDOCS

Fig. 4의 아키텍처 패턴을 GDOCS 시스템의 기능을 정확히 반영하고 오픈 소스 사용을 지원하는 관점에서 "Virtual Machine" 컴포넌트를 "Input Monitoring" 컴포넌트와 "Home Automation System Connection" 컴포넌트로 분리하여 정의하였다.

5.5 인스턴스 생성

Fig. 5에 주어진 조정된 아키텍처 패턴을 기반으로 패턴의 인스턴스를 생성함으로써, GDOCS 시스템에 대한 소프트웨어 아키텍처를 생성할 수 있다. 즉 "Non-Performance Computation" 컴포넌트의 인스턴스는 "Raising/Downing Door" 인스턴스로, "Performance-Critical Computation"은 도어가 하강하는 동안 장애물과 접촉하면 즉시 도어를 올려야 하는 기능을 위한 "Obstacle Detection" 인스턴스로, 그리고 "Input Monitoring"은 사용자나 홈 오토메이션에 입력되는 정보를 탐지하여 요청한 동작을 지시하는 "Sensor/Actuator" 인스턴스로 생성하였다. Fig. 6은 GDOCS 시스템에 대하여 생성된 인스턴스(즉, Child Module)을 나타낸다.

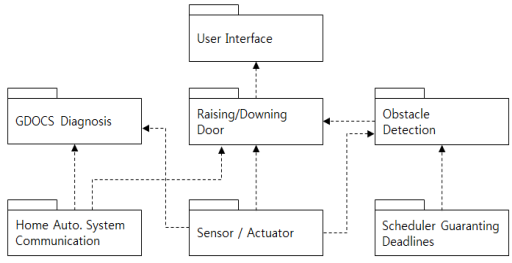

Fig. 6. (Draft) Software Architecture for GDOCS

Fig. 6에 나타난 인스턴스들은 오픈 소스의 재사용을 염두에 두고 생성된 아키텍처 이다. 즉 조정된 패턴으로부터 인스턴스를 생성할 때, 재사용 가능한 오픈 소스를 고려한다면, 해당 컴포넌트를 분할하거나 통합함으로써, 인스턴스를 생성할 수 있다. 여기서 통합이라함은 컴포넌트의 내부 로직 통합이 아닌 패키지 개념의 통합을 의미한다.

5.6 기능 할당

ADD 방법에 의한 소프트웨어 아키텍처 설계과정에서 후반부에 진행되는 작업은 각각의 아키텍처 컴포넌트에 기능을 배정하는 것과 기능간의 인터페이스를 정의하는 것이다. Fig. 6의 소프트웨어 아키텍처에 대하여 GDOCS 시스템의 기능을 할당하면 Table 2와 같다.

Table 2의 오른쪽 컬럼은 OSS(Open Source Software)의 사용 여부를 표시하는 필드로써, 코드 리버싱 과정에서 확보한, 또는 아키텍처링 과정에서 선정된 코드의 재사용이 가능한 컴포넌트이다.

Table 2. Allocation of Functionality to Architectural Components and OSS Assignments

Components	Allocated functionality	OSS
User Interface	- Door Open Button - Door Close Button	×
Raising/ Downing	- Motor on to raise - Motor on reversely to down - Stop motor	O
Obstacle Detection	- Detect object during door downing - Send interrupt for detection	O
GDOCS Diagnosis	- Request Built-In-Test run - Receive data from devices	
Sensor/ Actuator	- monitoring sensors, periodically - decision for input signals - Activation the corresponding device (actuator)	×
HAS Comm.	- Send/Receive message	×
Scheduler with Deadlines	- Task Scheduling	O

5.7 아키텍처 뷰 모델링

아키텍처 뷰 모델링은 소프트웨어 시스템이 갖는 다양한 특성을 고려하여 Information View, Concurrency View, Deployment View 등과 같은 다양한 측면에서 시스템의 아키텍처를 도출하는 것이다. Fig. 7은 GDOCS 시스템의 Concurrency View를 나타낸다.

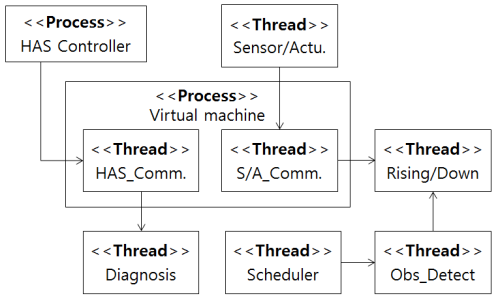

Fig. 7. Concurrency View for GDOCS Architecture

6. 사례 연구

본 절에서는 앞서 설명한 코드 리버싱 단계와 아키텍처 설계 단계에 대한 예제 시스템 적용을 통하여 제시한 SADOSS 기법의 적용성을 확인하였다. 본 절에서 적용하는 예제 시스템은 소프트웨어 아키텍처 등을 개발하기 위한 "UML 모델러" 소프트웨어이다.

6.1 예제 시스템의 요구사항

소프트웨어 아키텍처를 작성할 수 있는 UML 모델러에 대한 기본적인 요구사항은 다음과 같다.

- 클래스 다이어그램, 패키지 다이어그램, 컴포넌트다이어그램, 배치 다이어그램과 같은 기본적인 UML 다이어그램을 작성할 수 있어야 한다.
- 각 다이어그램의 심볼들을 각각 별도의 팔레트로 제공해야 한다.
- 다이어그램간의 정보 일관성을 보장할 수 있어야 한다.
- 오픈소스를 사용하여 기능 구현이 가능하도록 한다.

6.2 코드 리버싱을 통한 설계 정보 추출

6.1절에서 제시한 요구사항을 기반으로 UML 모델러를 개발하기 위하여 각 다이어그램의 요소, 즉 도형을 그리는 부분을 기존의 오픈 소스를 이용하고자 한다. 오픈 소스 사이트를 통해 Fig. 8과 같은 JAVA 오픈 소스 샘플 "Shape.java" 파일을 구한다. 이중 일부를 Fig. 8에 나타내었다.

```
 /** "Implementor" */
interface DrawingAPI {
    public void drawCircle(final double x, final double
y, final double radius);
}

/** "ConcreteImplementor"  1/2 */
class DrawingAPI1 implements DrawingAPI {
    public void drawCircle(final double x, final double
y, final double radius) {
        System.out.printf("API1.circle at %f:%f radius
%f₩n", x, y, radius);
    }
}

/** "ConcreteImplementor" 2/2 */
class DrawingAPI2 implements DrawingAPI {
    public void drawCircle(final double x, final double
y, final double radius) {
        System.out.printf("API2.circle at %f:%f radius
%f₩n", x, y, radius);
    }
}

/** "Abstraction" */
abstract class Shape {
    protected DrawingAPI drawingAPI;
    protected Shape(final DrawingAPI drawingAPI){
        this.drawingAPI = drawingAPI;
    }
    public abstract void draw();  // low-level
```

```
    public abstract void resizeByPercentage(final
double pct);                          // high-level
}

/** "Refined Abstraction" */
class CircleShape extends Shape {
    private double x, y, radius;
    public CircleShape(final double x, final double y,
final double radius, final DrawingAPI drawingAPI) {
        super(drawingAPI);
        this.x = x;  this.y = y;  this.radius = radius;
    }

    // low-level i.e. Implementation specific
    public void draw() {
        drawingAPI.drawCircle(x, y, radius);
    }
    // high-level i.e. Abstraction specific
    public void resizeByPercentage(final double pct) {
        radius *= (1.0 + pct/100.0);
    }
}

/** "Client" */
class DrawingShape {
    public static void main(final String[] args) {
        Shape[] shapes = new Shape[] {
            new CircleShape(1, 2, 3, new
DrawingAPI1()),
            new CircleShape(5, 7, 11, new
DrawingAPI2())
        };

        for (Shape shape : shapes) {
            shape.resizeByPercentage(2.5);
            shape.draw();
        }
    }
}
```

Fig. 8. A part of Source Code in "Shape.java"

Fig. 8의 소스 코드에 대하여 Doxygen의 산출물은 Fig. 9와 같으며, 일부 상세화 작업을 통하여 얻은 설계 정보는 Table 3과 같다.

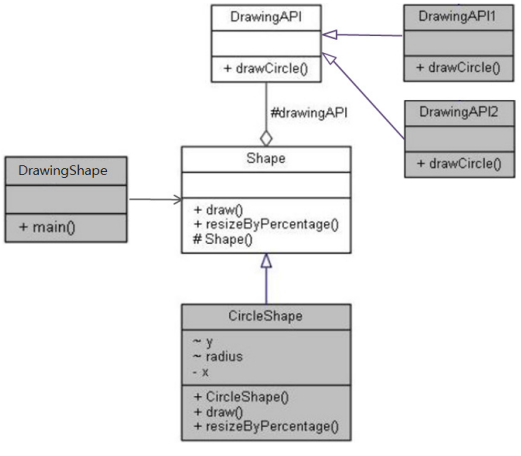

Fig. 9. Design Information: Collaboration diagram on "Shape.java"

Fig. 9로부터 앞서 정의한 코드 리버싱단계의 CR-3 스텝에 따라 추가적인 설계 정보를 도출한다. 특히 패턴의 이름(Pattern Name)과 의도(Intent), 그리고 동기(Motivation)에 대한 정보는 Doxygen 도구를 통해 얻은 Fig. 9의 다이어그램으로부터 쉽게 추출될 수 있었다.

Table 3. Design Information on Source "Shape.java"

SECTION	DESIGN INFORMATION
Pattern Name & Classification	Bridge Pattern (by DrawingShape Class)
Intent	Decouple modules from implementation to preserve independent behaviors
Also Known As	N/A
Motivation (Forces)	Apply design pattern concept to improve modularity and flexibility
Applicability	Drawing for rectangle, circle, etc.
Structure	Base classes: DrawingShape, Shape, DrawingAPI, Drived classes: CircleShape, DrawingApI1, DrawingAPI2 (Fig. 9)
Participants	Shape class
Consequences	N/A
Implementation	General Classes using Java
Sample Code	Fig. 8
Known Uses	N/A
Related Patterns	N/A

6.3 예제 시스템 아키텍처 설계

코드 리버싱 단계의 산출물을 기반으로 아키텍처를 설계하기 위하여 먼저 아키텍처 드라이버와 설계 전술을 고려해야 한다. UML 모델러 개발을 위하여 이들을 다음과 같이 정의 하였다.

[아키텍처 드라이버]

- 모델 요소가 캔버스(Canvas)에 그려졌을 때, 즉시 플러싱(Flushing)되어야 한다.
- 모델 요소가 추가될 때, 다이어그램간의 일관성 체크가 이루어져야 한다.
- 오픈 소스를 활용할 수 있는 구조로 제시되어야 한다.

[아키텍처 전술]

앞서 정의한 드라이버 중에서 오픈 소스 재사용에 대한 아키텍처 전술은 Fig. 10과 같다.

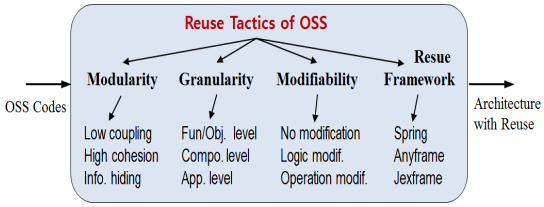

Fig. 10. Architectural Drivers for OSS Reuse

정의된 드라이버와 아키텍처 전술 중 컴포넌트 수준의 재사용 Granularity, No modification, 스프링 프레임워크를 기준으로 하는 UML 모델러의 소프트웨어 아키텍처는 Fig. 11과 같다.

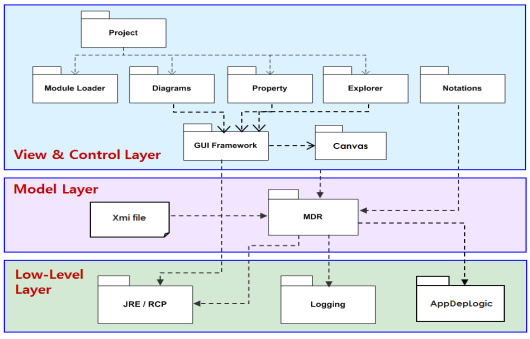

Fig. 11. Architecture for UML Modeler

6.4 적용성 분석

본 논문에서 제시하는 SADOSS 방법의 유용성을 평가하기 위하여 다음과 같은 4가지 질문에 대하여 살펴보았다.

Q1: 아키텍처의 융통성이 유지 되는가?
Q2: 오픈 소스 재사용이 가능한가?
Q3: 오픈 소스의 설계 정보가 유용한가?
Q4: 제시한 SADOSS 방법의 장점은?

6.4.1 아키텍처 융통성의 보전

소프트웨어 아키텍처를 설계할 때, 아키텍처를 구성하는 요소들은 가능한 독립적인 컴포넌트로 구성될 수 있도록 하며, 또한 변경의 영향을 최소화하는 방향으로 설계된다. 본 논문의 SADOSS 방법에 따라 설계된 아키텍처가 이러한 컴포넌트 독립성과 변경 영향의 최소화(즉, 모듈성)를 충분히 지원할 수 있는지의 관점은 매우 중요하다.

(1) 첫 번째 관점에서 보면 재사용 가능한 오픈 소스들은 대체적으로 정제된 API를 통해서 독립적인 기능을 수행할 수 있도록 개발된다. 이러한 특성을 고려할 때, 오픈 소스 재사용이 아키텍처의 특성을 저해하지 않는다고 할 수 있다.

(2) 선정된 오픈 소스의 재사용을 고려하여 아키텍처 구성요소를 확정할 때, 오픈 소스의 설계정보를 기반으로 컴포넌트를 분리 또는 통합할 수 있다. 아키텍처 컴포넌트의 분리는 사실상 발생하기 쉽지 않은 경우이지만, 분리된 두 개의 서브 컴포넌트가 다시 하나의 상위 컴포넌트로 추상화될 수 있기 때문에 기본적인 아키텍처 구조에는 변함이 없다.

아키텍처 구성요소가 통합되는 경우에는 각 컴포넌트가 독립성과 모듈성이 고려되어 식별되었기 때문에 통합된 컴포넌트의 독립성과 모듈성에는 큰 차이가 없다.

6.4.2 오픈 소스 재사용 가능성

두 번째 질문은 오픈 소스를 고려한 아키텍처를 설계한 경우, 선정된 오픈 소스의 수정없이 재사용 가능한가에 대한 관점이다. 선정된 오픈소스의 세부 로직 또는 연산에 대한 수정이 이루어진다면, SADOSS 방법의 장점은 많이 감소될 것이다.

(1) 오픈 소스의 재사용은 일반적으로 API의 호출을 통해 이루어지기 때문에 Application의 인터페이스 Contracts의 정보를 기반으로 아키텍처 컴포넌트의 관계가 정의될 수 있다. 따라서 인터페이스의 변화가 없다면 선정된 오픈 소스를 변경없이 재사용할 수 있다.

(2) 오픈 소스를 변경하지 않고 사용하는 또 다른 방법은 아키텍처 컴포넌트에 대한 Wrapper를 정의하는 방법이다. 이는 선정된 오픈 소스의 인터페이스 변경이 필요하더라도 Wrapper를 통한 변경없는 재사용이 가능해진다.

6.4.3 오픈 소스 설계정보의 유용성

재사용 가능한 오픈 소스에 대하여 설계 정보 복구를 통해 재사용을 확정하게 된다. 이 과정에서 오픈 소스에 대한 분석 및 이해가 이루어지고, 코드의 작성 의도 및 활용법도 이해하게 된다.

(1) 제안하는 SADOSS 방법에서는 GoF에서 제안하

는 패턴 명세 템플릿을 이용하여 오픈 소스에 대한 설계 정보를 복구한다. 이 템플릿을 이용하면 오픈 소스의 코드 구조에 대한 단순한 이해가 아니라 코드가 갖는 철학 및 의도를 파악하는데 도움이 된다.

(2) 아키텍처를 설계하는 과정에서 오픈소스의 설계 정보는 아키텍처 컴포넌트의 기능, 크기, 모듈화 등을 결정하는데 사용된다.

(3) 제시한 SADOSS 방법은 오픈 소스의 재사용을 고려하여 소프트웨어 아키텍처를 개발하는 방법이기 때문에 이 방법에 의해 아키텍처가 설계되었다면 선정된 오픈 소스는 반드시 재사용될 것이다.

6.4.4 SADOSS 방법의 장점

본 논문에서 제시하는 SADOSS 방법 개발의 근본 취지는 오픈 소스의 규모가 날로 방대해지고 있으며, 또한 매우 다양한 영역에서 사용되어지고 있기 때문이다[20].

오픈 소스를 통한 소프트웨어 시스템의 개발이 매우 신속하고 안정적인, 그리고 적은 비용에 의한 소프트웨어 개발을 가능하게 하기 때문에 본 연구에서는 이러한 오픈 소스 재사용의 장점을 이용하고자 하였다.

다만, 소프트웨어의 수명주기가 짧아지고, 사회, 기술, 정책의 급속한 변화 및 시장의 요구의 다양성 및 신규성을 충족하기 위하여 보다 안정적이고 확장성이 높은 소프트웨어 아키텍처를 설계하는 것은 조직의 비즈니스를 위한 매우 중요한 사항이다[21]. 따라서 본 연구에서 제시하는 SADOSS 방법의 장점은 다음과 같이 정리할 수 있다.

(1) 개발 비용의 절감 : 소프트웨어 시스템의 개발 비용에 대하여 오픈 소스 재사용으로 비용 절감이 가능하다. 비록 오픈 소스에 대한 설계 정보의 복구를 위한 노력이 요구되기는 하지만, 이들은 Doxygen, Bunch 와 같은 도구들에 의해 자동화될 수 있다.

(2) 오픈 소스의 재사용 : 재사용 대상의 후보 오픈 소스에 대한 설계 정보 복구과정을 통하여 선정된 오픈 소스에 대한 특성을 보다 정확히 이해할 수 있다. 이는 재사용 과정에서 보다 정확한 코드의 이해 및 활용 의도에 맞게 오픈 소스를 활용할 수 있도록 한다.

(3) 아키텍처 시뮬레이션 : 오픈 소스 재사용을 지원하는 아키텍처 개발 시, 아키텍처 시뮬레이션을 보다 용이하게 수행할 수 있다. 특히 재사용되는 아키텍처 컴포넌트에 대해서는 별도의 시뮬레이션 Harness를 개발할 필요가 없이 수행될 수 있다.

(4) 아키텍처 및 산출물 문서화 : 소프트웨어 개발과정에서 작성해야 하는 문서화 노력이 절감될 수 있다. 오픈 소스의 설계 복구를 통해 부분적인 문서화가 가능하기 때문에 본 제안 방법은 개발 산출물에 대한 문서화 비용을 절감할 수 있다는 장점이 있다.

7. 결론 및 향후 연구

오픈 소스에 대한 활용 분야가 넓어지고, 그 유용성이 증가됨에 따라, 오픈 소스 기반 개발이 일반화되고 있다. 기업의 비즈니스 시스템 개발에서 소프트웨어 아키텍처는 미래의 소프트웨어 비즈니스에 대응하고, 안정적인 시스템 운영 및 기능 확장의 융통성을 제공한다는 측면에서 유용하다. 본 연구에서는 오픈 소스 활용을 고려한 아키텍처 설계 방법, SADOSS를 제안한다. 제안하는 방법을 통한 아키텍처 개발은 오픈 소스 활용을 통해 개발 비용의 절감은 물론 개발되는 소프트웨어 시스템의 품질도 함께 제공할 수 있다는 장점을 제공한다.

SADOSS 방법에 대한 향후의 연구 내용은 아키텍처 설계를 통해 아키텍처 구성 요소에 대한 명세가 이루어지면, 이를 기반으로 오픈 소스를 자동으로 탐색하여 매핑해주는 기법에 대하여 연구하고자 한다.

ACKNOWLEDGMENTS

이 논문은 정부(미래창조과학부)의 재원으로 한국연구재단-차세대 정보컴퓨팅 기술개발사업(No.NRF-2015M3C4A7030505)의 지원을 받아 수행한 것임.

REFERENCES

[1] R. Kapur, et al., *Open Source Development: Ideal for application development and administrators*, DB2 On Campus Book Series, IBM, July 2010.

[2] C. Kramer and L. Prechelt, “Design Recovery by Automated Search for Structural Design Patterns in

Object-Oriented Software," *Proceedings of the Third Working Conference on Reverse Engineering*, pp. 1-8, 1996. DOI: 10.1109/WCRE.1996.558905

[3] J. Niere, J. P. Wadsack and L. Wendehals, *Design Pattern Recovery Based on Source Code Analysis with Fuzzy Logic*, TR0RI-01-222, University of Paderborn, Germany, 2001.

[4] N. R. Carvalho, A. Simoes and J. Almeida, "DMOSS: Open Source SW Documentation Assessment," *Computer Science and Information Systems*, Vol. 11, No. 4, pp. 1191 - 1207, 2014. DOI: 10.2298/CSIS131005027C

[5] Find, "Create, and Publish Open Source Software for Free," https://sourceforge.net, 2017. 3.

[6] Gifhub, "How people build software," https://github.com, 2017. 2.

[7] E. Constantinou, G. Kakarontzas and I. Stameloos, "Open Source Software: How Can Design Metrics Facilitate Architecture Recovery?," *Proceeding of the 4th Workshop on Intelligent Techniques in Software Engineering*, pp. 111-222, 2011

[8] L. Heinemann, F. Deissenboeck, M. Gleirscher, B. Hummel and M. Irlbeck "On the Extent and Nature of Software Reuse in Open Source Java Projects", *Proceeding of the International Conference on Software Reuse 2011, Lecture Notes in Computer Science, Vol 6727. Springer*, pp. 207-222, 2011. DOI: 10.1007/978-3-642-21347-2_16

[9] Doxygen, http://www.doxygen.org, 2017. 2.

[10] S. Mancoridis, B. S. Mitchell, Y. Chen and E. R. Gansner, "Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures," *Proceeding of the International Conferencce on Software Maintenamce (ICSM'99)*, pp. 1-10, 1999.

[11] G. Rasool, P. Meader and I. Philippow, "Evaluation of design pattern recovery tools," *Procedia Computer Science*, Vol. 3, pp. 813 - 819, 2011

[12] Y. Cai, H. Wong, S. Wong and L. Wang. "Leveraging design rules to improve software architecture recovery," *Proceeding of the International ACM Sigsoft Conference on the Quality of Software Architectures*, pp. 133 - 142, 2013.

[13] H. Ryu and W. J. Lee, "A Study on UML based Modeling and Automatic Code Generation," *Journal of Convergence for Information Technology*, Vol. 2, No. 1, pp. 33-40, 2012.

[14] J. S. Park, J. S. Kwon, J. E. Hong and M. Choi, "Software Architecture Recovery for Android Application Reuse," *Journal of Convergence Society for SMB*, Vol. 3, No. 2, pp. 9-17, Jun. 2013.

[15] Y. Choi, D. Kim and J. E. Hong, "An Extraction Technique of Design Pattern for Enhancing Reusability and Extendibility of JAVA Open sources," *Proceeding of the KIISE Conference on Software Engineering (KCSE 2017)*, Vol. 19, No. 1, pp. 203-210, 2017.

[16] R. Wojcik, F. Bachmann, L. Bass, P. Clements, P. Merson, R. Nord and B. Wood, *Attribute-Driven Design (ADD), Version 2.0*, CMU/SEI-2006-TR-023, SEI CMU, Nov. 2006.

[17] F. Bachmann, L. Bass and M. Klein, *Deriving Architectural Tactics: A Step Toward Methodical Architectural Design*, CMU/SEI-2003-TR-004, SEI CMU, Mar. 2003

[18] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad and M. Stal, *Pattern-Oriented Software Architecture*, Volumn 1: A System of Patterns, Wiley, 1996.

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

[20] Rachel Roumeliotis, *5 software development trends shaping enterprise*, O'Reilly, Jan., 2017

[21] H. Cervantes, R. Kazman, *Designing Software Architectures: A Practical Approach*, Addison-Wesley, 2016.

저 자 소 개

최 용 석(Yongseok Choi) [학생 회원]

- 2010년 2월 : 한국기술교육대학교 학사
- 2016년 3월 : 충북대학교 컴퓨터과학과 석사과정 입학
- 2016년 3월 ~ 현재 : 충북대학교 컴퓨터과학 석사과정

<관심분야> : 소프트웨어공학, 코드 리팩토링, 소프트웨어 설계 패턴, 사물 인터넷(IoT)

홍 장 의(Jang-Eui Hong) [종신회원]

- 2001년 2월 : 카이스트 전산학과 (전산학박사)
- 2002년 10월 : 국방과학연구소, 선임연구원
- 2002년 11월 ~ 2004년 8월 : ㈜ 솔루션링크 부설기술연구소장
- 2004년 9월 ~ 현재 : 충북대학교 소프트웨어학과 교수

<관심분야> : 모델 기반 소프트웨어공학, 소프트웨어 품질, 소프트웨어 아키텍처, 저전력 소프트웨어, CPS