論文
3-1-13

XML 기반 소프트웨어 아키텍처 재구성과 명세 모델 (XML based Software Architecture Restructuring and Specification Model)

박 근 순*
(Keun-Soon Park)

요 약

소프트웨어 개발에 적합한 설계는 하나 이상 존재할 수 있고, 소프트웨어 아키텍처는 설계 과정 동안 수정, 갱신, 대체로 인해 변경될 수 있으므로, 다양한 관점에서 설계된 아키텍처의 서로 다른 버전들을 유지 관리할 수 있는 방법과, 아키텍처의 변경 요소들을 효율적으로 명세할 수 있는 방법이 요구된다.

이 연구에서는 기존에 이미 정의되어 있는 아키텍처와, 이를 재사용하여 설계한 새로운 아키텍처를 통합된 구조의 소프트웨어 아키텍처로 재구성할 수 있는 방법과, 재구성 아키텍처 명세에 필요한 명세 요소와 명세 구조에 대해 정의하고 있다. 제안된 통합된 구조의 명세 방법은 아키텍처 설계 과정에서 다양한 개발자 관점에서 정의된 아키텍처들을 참조 및 재사용할 수 있는 방법을 제공하고, 아키텍처의 수정, 대체, 갱신과 같은 변경 요소에 대해 유연성 있는 명세 방법을 제공한다.

ABSTRACT

In software development process, appropriate design can exist more than one. And software architectures can be restructured by modification and update during design processes. Therefore, the method that specify efficiently the modification elements of architectures, and manage different versions of architecture designed through various aspects is required.

In this study, we propose software architecture restructuring mechanism that can restructure legacy architecture and a new software architecture designed with reuse of it in integrated form, and define specification elements and structure of the proposed restructured architecture specification. It provides the method that can reference and reuse architectures designed with various aspects of developers in architecture design processes. In addition, it supports flexible specification method for specification of modification elements such as revision, substitution, update of legacy software architecture.

1. 서론

전체적인 시스템을 컴포넌트의 조합으로 구성하는 소프트웨어 아키텍처는 설계 단계에서 전체적인 시스템의 문제점 파악을 위해, 그리고 시스템의 요구사항과 실제 구현 사이에 있을 수 있는 차이점을 완화시킬 수 있는 중요한 설계 요소로 인식되고 있다. 이는 새로운 어플리케이션 개발 비용을 감소하고, 서로 밀접하게 관련된 다양한 소프트웨어 제품군 간의 일반성을 향상시킬 수 있는 방법을 제공한다[1, 2].

* 정회원 : 인하공업전문대학 컴퓨터정보공학부 교수

논문접수 : 2002. 1. 15.
심사완료 : 2002. 1. 24.

소프트웨어 아키텍쳐는 설계 과정 동안 수정, 갱신, 대체로 인해 변경될 수 있고, 새로운 소프트웨어 개발에 적합한 설계는 하나 이상 존재할 수 있다. 따라서 개발자는 다양한 관점에서 설계된 아키텍쳐들의 서로 다른 버전들을 유지 관리할 수 있는 방법과, 아키텍쳐의 변경 요소들을 효율적으로 명세할 수 있는 방법이 고려되어야 한다[3].

기존의 다양한 아키텍쳐 명세 언어들은 아키텍쳐 명세를 위한 정형화된 명세 요소들과 분석 방법을 제공함으로써, 시스템의 전체적인 구조를 명세할 수 있는 방법을 제공한다[4]. 하지만 정형화된 명세 언어들은 아키텍쳐 명세 구조 자체가 복잡하고, 수정과 대체로 인한 아키텍쳐의 변화에 유연성 있는 명세 방법을 제공하기 어렵다.

이 연구에서는 기존에 이미 정의되어 있는 아키텍쳐와, 이를 재사용하여 설계한 새로운 아키텍쳐를 통합된 구조의 소프트웨어 아키텍쳐로 재구성할 수 있는 방법을 제안하고 있다. 그리고, 통합된 구조의 소프트웨어 아키텍쳐 명세에 필요한 명세 요소와 명세 구조에 대해 정의하고 있다.

제안된 소프트웨어 아키텍쳐 재구성 방법은 아키텍쳐 설계 과정에서 다양한 개발자 관점에서 정의된 아키텍쳐들을 참조 및 재사용할 수 있는 방법을 제공하고 있다. 그리고 서로 관련된 아키텍쳐들을 통합된 구조로 재구성하여 저장함으로써 아키텍쳐 명세 관리를 위한 기본적인 분류를 제공한다.

XML 기반 소프트웨어 아키텍쳐 명세 구조는 기존의 아키텍쳐 명세 언어의 명세 요소들을 대부분 지원하면서 아키텍쳐의 재사용과 수정, 대체, 갱신과 같은 아키텍쳐의 변경 요소에 대해 유연성 있는 명세 방법을 제공한다. 그리고, 기존의 명세 언어와는 달리 비정형적인 방법에 의한 명세 방법을 제공하므로 보다 쉬운 형태의 아키텍쳐 명세가 가능하다.

2. 소프트웨어 아키텍쳐 명세와 버전 모델

2.1 소프트웨어 아키텍쳐 명세를 위한 명세 요소

소프트웨어 아키텍쳐에 대한 연구는 새로운 어플리케이션 개발 비용을 감소하고, 서로 밀접하게 관련된 다양한 소프트웨어 제품군 간의 일반성을 향상시킬 수 있는 방향으로 진행되어 왔다. 일반적인 소프트웨어 아키텍쳐 특성에 기반한 새로운 소프트웨어 개발은 아키텍쳐를 구성하는 컴포넌트와 커넥터들의 전체적인 연결 구조과 통신 구조를 중심으로 아키텍쳐를 명세할 수 있는 방법을 필요로 한다[5]. 아키텍쳐 명세를 위해서는 정형화된 명세 요소가 필요하고, 이러한 요소들을 분석하여 소프트웨어 아키텍쳐를 명세할 수 있어야 한다. 소프트웨어 아키텍쳐 명세 언어는 명세를 위한 정형화된 방법을 제공하고 있고, 응용의 구현보다는 고 수준에서의 전체적인 응용 구조를 정의하는데 중점을 두고 있다[6].

특정 영역(domain) 내에서, 그리고 일반적인 목적에서의 소프트웨어 아키텍쳐를 명세하기 위해 기존에 ACME, Aesop, C2, Darwin, MetaH, Rapide, SADL, UniCon, Weaves, Wright 등과 같은 많은 아키텍쳐 명세 언어가 이미 제시되어 사용되고 있다. 이러한 아키텍쳐 명세 언어들은 아키텍쳐를 구성하는 컴포넌트와 커넥터 단위에서의 명세 요소들을 정의하고 있다[4]. 컴포넌트와 커넥터에 대한 명세 요소로는 인터페이스(interface), 타입(type), 의미 정보(semantics), 제약조건(constraints), 비함수적 속성(non functional properties) 등이 있는데, 소프트웨어 아키텍쳐를 명세하기 위해서는 아키텍쳐를 구성하고 있는 컴포넌트와 커넥터 단위의 이러한 명세 요소들을 고려하여야 한다[4, 7].

2.2 버전 모델과 버전화 방법

소프트웨어의 설계는 공학적인 산물로써, 다양한 컴포넌트들의 복잡한 설계를 관리하는 측면을 포함하고 있다. 각각의 컴포넌트는 설계 과정동안 계속해서 발전하는 형태를 취하는데, 이러한 설계 과정에서의 발전성 때문에 다른 단계에서 적용될 수 있는 컴포넌트의 다른 버전들을 유지 관리할 필요가 있다. 그리고 적합한 컴포넌트에 대한 설계는 하나 이상 존재할 수 있으므로, 자신의 개발 환경에 적합한 컴포넌트 버전을 적용할 수 있는 방법 또한 필요하다[3].

버전 모델은 버전화에 필요한 항목들을 정의하고, 하나의 항목에 대한 모든 버전이 공유 가능한 일반적인 속성들 뿐만 아니라 버전들간의 차이점에 대해

서 정의한다[3]. 버전 모델은 이러한 버전 요소들의 집합을 구성할 수 있는 방법 또한 결정한다. 이러한 목적을 위해 버전 모델은 수정(revisions)과 변형(variants)과 같은 버전 작업의 전개 방향에 대해 소개하고, 하나의 버전이 모델이 표현하는 상태나 기본 요소와 관련된 변화에 의해 특징지어 질 수 있는지 여부를 정의한다[8].

이 연구에서는 기존 아키텍쳐의 재사용 과정에서 아키텍쳐를 구성하는 컴포넌트와 커넥터를 대체하는 방식으로 새로운 아키텍쳐 설계가 가능하다는 점에서 대체값에 의한 버전화 방법인 대체 버전화 개념을 고려한다. 그리고 아키텍쳐의 변경 요소에 대한 명세는 특정 컴포넌트와 커넥터 명세에 대해 이루어질 수 있다는 점에서 원자 객체들을 사용한 버전 제어 방식인 컴포넌트 버전화 방법과, 컴포넌트 버전화를 일반화하는 구조적 명세의 모든 계층에 대한 버전화 방법인 전체 버전화 방법 또한 고려한다.

3. 소프트웨어 아키텍쳐 재구성과 명세

3.1 통합된 구조의 소프트웨어 아키텍쳐 명세

기존의 소프트웨어 아키텍쳐에 대한 재사용을 위해 개발자는 기존 아키텍쳐에 특정한 구조를 추가, 삭제 및 대체함으로써 새로운 아키텍쳐를 구성할 수 있다. 이 연구에서는 소프트웨어 아키텍쳐 설계 과정에서 재사용되는 아키텍쳐의 구성을 그대로 유지한 상태에서 새로 설계된 아키텍쳐의 구성을 통합된 구조로 재구성할 수 있는 메커니즘을 제안한다.

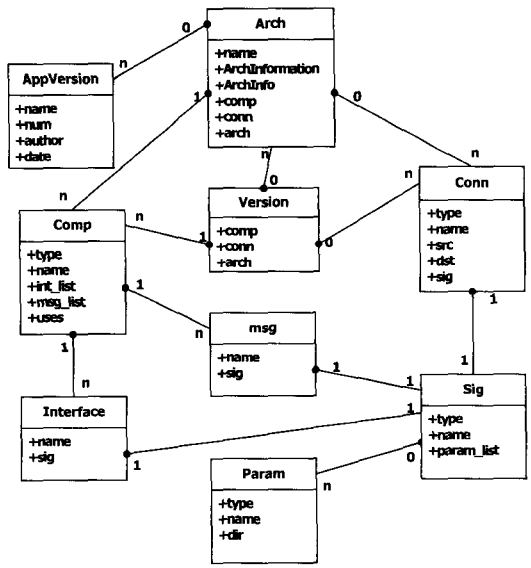
이러한 통합된 구조의 소프트웨어 아키텍쳐를 저장소에 효율적으로 저장하고 관리하기 위해서는 우선 통합된 구조로 재구성된 소프트웨어 아키텍쳐의 명세에 필요한 명세 요소와 명세 구조에 대한 정의가 필요하다. 이를 위해 재구성 소프트웨어 아키텍쳐가 포함하고 있는 각각의 아키텍쳐에 대한 전체적인 정보가 아키텍쳐 명세에 정의되어야 하고, 포함된 각각의 아키텍쳐를 표현하기 위한 정보들이 추가적으로 정의되어야 한다.

<리스트 1>은 이 연구에서 제안하는 통합된 구조의 재구성 소프트웨어 아키텍쳐를 XML 기반으로 명세하기 위한 DTD를 나타내고 있다.

<리스트 1> XML 기반 아키텍쳐 명세의 DTD
<List 1> DTD for XML based architecture specification

```
<?xml version="1.0"?>
<!ELEMENT arch ((version+, comp+, conn*, arch*)|(name, AppVersion*, comp+, conn*, arch*))>
<!ELEMENT comp ((version+)|(type?, name, int_list?, msg_list?))>
<!ELEMENT conn ((version+)|(type?, name, src, dst, sig?))>
<!ELEMENT version ((type?, name, src, dst, sig?)|(name, comp+, conn*,arch*)|(name, int_list?, msg_list?))>
<!ATTLIST AppVersion name CDATA #REQUIRED>
<!ATTLIST AppVersion number CDATA #REQUIRED>
<!ATTLIST AppVersion author CDATA #REQUIRED>
<!ATTLIST AppVersion date CDATA #REQUIRED>
<!ATTLIST version name CDATA #IMPLIED>
<!ELEMENT int_list (num, interface*)>
<!ELEMENT msg_list (num, msg*)>
<!ELEMENT interface (name?, sig)>
<!ELEMENT msg (name?, sig)>
<!ELEMENT sig (name?, type?, param_list)>
<!ELEMENT param_list (num, param*)>
<!ELEMENT param (type?, name?, dir?)>
```

정의한 DTD를 기반으로 하는 XML 기반 소프트웨어 아키텍쳐 명세 구조는 [그림 1]을 통해 클래스 다이어그램으로 표현하였다.



[그림 1] 아키텍쳐 명세 구조 클래스 다이어그램
[Fig. 1] Class Diagram for Architecture Specification Structure

개발자들의 다양한 관점에서 설계된 아키텍쳐들을 모두 포함하는 통합된 구조의 소프트웨어 아키텍쳐 명세에서 아키텍쳐 이름과 개발자 이름과 같은 각각의 아키텍쳐 설계 정보를 표현하기 위한 AppVersion 클래스가 아키텍쳐 명세의 최상위에 포함되어 있으며, 컴포넌트와 커넥터로 구성되는 아키텍쳐에는 Version클래스를 추가하여 개발자들의 서로 다른 관점에서의 아키텍쳐 정의를 모두 반영할 수 있도록 정의하고 있다. Version클래스는 컴포넌트 형태의 하부 아키텍쳐를 포함할 수 있으므로 Arch 클래스의 컴포넌트와 커넥터 클래스를 포함하고 있는 형태를 가진다.

3.2 소프트웨어 아키텍쳐 재구성

소프트웨어 아키텍쳐 설계 과정에서 개발자는 기존에 이미 정의되어 있는 아키텍쳐들을 재사용하여 자신의 개발 환경에 적합한 새로운 소프트웨어 아키텍쳐를 설계할 수 있다. 이 연구에서는 이러한 설계 과정에서 새로 구성된 아키텍쳐를 기존의 소프트웨어 아키텍쳐 설계를 그대로 유지하는 상태에서 저장소에 반영하기 위해 통합된 구조의 재구성 소프트웨어 아키텍쳐 명세 구조를 정의하고 있다.

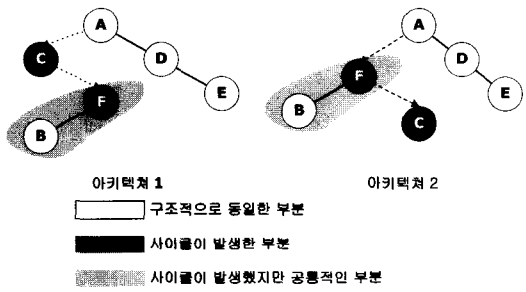
기존에 정의되어 있는 소프트웨어 아키텍쳐를 재사용하여 새로운 아키텍쳐를 설계하고, 이를 기존의 아키텍쳐와 통합된 구조로 관리하기 위해서는 우선 두 개의 아키텍쳐에서 공통적으로 정의된 부분과 서로 다른 관점에서 정의된 부분을 추출하여 이를 통합된 구조로 표현할 수 있는 방법이 필요하다. 정의된 아키텍쳐들간 공통적인 정의 부분과 서로 다른 관점에서의 정의 부분에 대한 정보를 추출하는 작업은 통합된 구조의 아키텍쳐 명세에 있어 서로 다른 부분의 명세에만 각각의 아키텍쳐에 대한 정보를 추가하여 명세에 적용하기 위한 것이다.

이러한 아키텍쳐들간 서로 다르게 정의된 부분에 대해 추출한 정보를 기반으로 기존의 정의된 아키텍쳐와 아키텍쳐 설계 과정에서 개발자가 새로 설계한 아키텍쳐 사이의 공통된 관점과 서로 다른 관점으로 정의된 부분을 구분하여 아키텍쳐 명세에 적용함으로써, 하나의 통합된 구조의 소프트웨어 아키텍쳐를 명세할 수 있는 방법을 제공하고 있다.

아키텍쳐 설계 과정에서 개발자가 새로운 소프트웨어 아키텍쳐 명세를 정의하면, 기존의 소프트웨어 아키텍쳐 명세와 비교하여 공통 명세 부분과 서로 다른 관점을 구분하여 하나의 통합된 구조의 아키텍쳐 명세로 재구성하여야 한다. 소프트웨어 아키텍쳐 명세들의 서로 다른 관점에서의 정의 부분에 대한 정보 추출을 위해서는 하위 계층의 아키텍쳐를 포함하는 컴포넌트의 포함 관계와 계층간 포함 관계가 뒤바뀐 모든 컴포넌트들을 탐색해야 한다.

이 연구에서는 컴포넌트와 하위 계층 아키텍쳐간, 그리고 계층간의 포함 관계가 뒤바뀐 형태를 아키텍쳐의 사이클이라 정의하고, 이러한 사이클이 발생한 경우 사이클 발생 부분을 모두 명세에 반영하는 통합된 형태의 소프트웨어를 구성한다.

[그림 2]는 저장소의 기존 아키텍쳐와 개발자에 의해 정의된 새로운 아키텍쳐간 사이클이 발생한 예를 보여주고 있다. 그림에서 흰색으로 표시된 컴포넌트들은 기존 아키텍쳐와 공통된 관점에서 설계된 포함 관계가 일치하는 형태의 컴포넌트들을 나타내며, 짙은 회색으로 표현된 컴포넌트들은 설계된 아키텍쳐 명세 사이에서 사이클이 발생한 경우를 표현한다. 옅은 회색으로 표현된 컴포넌트들은 실제로 사이클이 발생한 부분에 포함되지만 컴포넌트간 관련성은 일치하는 부분을 나타낸다.

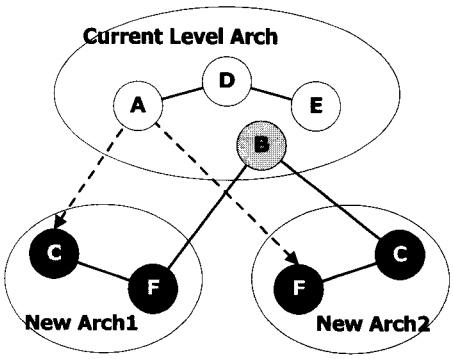


[그림 2] 아키텍쳐들간 사이클 발생 예

[Fig. 2] Cycle occurrence between two architectures

이 연구에서는 이러한 사이클이 발생했을 때, 사이클이 발생된 컴포넌트들을 서로 다른 평면으로 분리하고, 공통된 관점에서 정의된 컴포넌트들에 대한 연관 관계를 생성함으로써 기존 아키텍쳐와 새로 설계된 아키텍쳐를 모두 반영하는 통합된 형태의 소프트웨어 아키텍쳐를 명세한다.

[그림 3]은 [그림 4]에서 사이클이 발생한 컴포넌트를 나타내는 C, F를 서로 다른 평면으로 분리하여 기존 아키텍쳐와 새로 설계된 아키텍쳐를 모두 반영하고 있는 통합된 구조의 소프트웨어 아키텍쳐를 재구성한 예를 부여주고 있다.



[그림 3] 통합된 구조의 소프트웨어 아키텍쳐

[Fig. 3] Integrated structure of software architecture

[그림 3]에서 Current Level Arch와 New Arch1을 통해 아키텍쳐 1의 구조가 변형 없이 수용되었고, Current Level Arch와 New Arch2를 통해 아키텍쳐 2의 구조 또한 변형 없이 수용되어 통합된 구조의 소프트웨어 아키텍쳐로 재구성 되었음을 알 수 있다.

기존의 아키텍쳐와 개발자가 새로 정의한 아키텍쳐를 통합된 구조의 소프트웨어 아키텍쳐로 재구성하여 관리하는 방법은 아키텍쳐 단위의 계층간 포함 관계에서 사이클 발생 부분을 발견하여 적용 가능할 뿐만 아니라, 컴포넌트 단위와 커넥터 단위의 연관 관계에서 사이클을 발견하여 통합된 구조의 소프트웨어 아키텍쳐를 생성할 수도 있다.

이 연구에서 제안하는 서로 다른 개발자에 의해 정의된 소프트웨어 아키텍쳐를 구조별로 따로 제공하는 것이 아니라 하나의 통합된 구조로 제공하는 방법은 개발자가 새로운 소프트웨어 개발을 위한 아키텍쳐 설계 단계에서 다양한 관점에서 정의된 소프트웨어 아키텍쳐를 참조하여 재사용할 수 있도록 함으로써 기존의 아키텍쳐에 대한 재사용성과 개발에서의 유연성을 향상시킬 수 있음을 의미한다.

3.3 소프트웨어 아키텍쳐 재구성 모듈 설계

이 절에서는 저장소에 저장되어 있는 기존의 소프트웨어 아키텍쳐와 개발자가 아키텍쳐 설계 과정에서 이러한 저장소의 아키텍쳐를 재사용하여 새로 설계한 소프트웨어 아키텍쳐를 하나의 통합된 구조로 명세하는데 필요한 모듈들을 설계한다. 통합된 구조의 소프트웨어 아키텍쳐를 명세하기 위해서는 개발자의 새로운 아키텍쳐 설계를 XML 기반 아키텍쳐 명세 구조로 명세하는 모듈과 기존의 아키텍쳐 정의간 공통적인 관점에서 정의된 부분과 사이클 발생 부분에 대한 정보를 추출하는 모듈, 그리고 이러한 추출 정보를 기반으로 아키텍쳐들을 통합하는 모듈이 필요하다.

개발자가 저장소에 저장되어 있는 기존의 소프트웨어 아키텍쳐를 재사용하여 새로 설계한 소프트웨어 아키텍쳐는 전체적인 아키텍쳐 명세 모듈을 통해 XML 기반 아키텍쳐 명세 구조로 명세된다. XML 형태로 명세된 개발자의 새로운 소프트웨어 아키텍쳐는 기존의 소프트웨어 아키텍쳐 구성을 유지한 상태에서 통합된 구조의 소프트웨어 아키텍쳐를 재구성 하기 위해 아키텍쳐간 공통 부분과 사이클 발생 부분에 대한 정보를 추출하는 차이점 탐색 모듈로 입력된다. 차이점 탐색 모듈을 통해 추출된 아키텍쳐의 공통 부분과 사이클 발생 부분에 대한 정보를 기반으로 통합 모듈은 사이클이 발생한 아키텍쳐, 컴포넌트, 그리고 커넥터들을 다른 계층으로 분리시키고 서로간의 연관 관계와 포함 관계에 대한 정보를 생성함으로써 재구성된 소프트웨어 아키텍쳐가 하나의 통합된 구조로 명세될 수 있다.

<알고리즘 1>은 개발자에 의해 새로 설계된 소프트웨어 아키텍쳐 명세의 전체 구조에 대해 컴포넌트간 연관 관계와 계층간 포함 관계를 기존의 소프트웨어 아키텍쳐와 비교하여 사이클 발생 부분에 대한 정보를 추출하는 알고리즘을 나타낸 것이다.

<알고리즘 1> 아키텍쳐간 차이점 탐색 알고리즘

<Algorithm 1> Difference detection algorithm between architectures

method Difference Detection between Architectures

```
Query Architecture : Q
Retrieved Architectures : RA
Begin Find Difference Detection
  Arch integArch := RA.getFirstArch();
  while (integArch != NULL)
    ArchList Archs := integArch.getEachArch();
    Arch a := Archs.getFirstArch();
    While (a != NULL)
      CompList ArchComps :=
        a.getSimpComps();
      CompList QueryComps :
        Q.getSimpComps();
      ConnList ArchConns := a.getConns();
      ConnList QueryConns : = Q.getConns();
      For (i = 0; i < QueryComps.getSize(); i++)
        Comp comp :=
          ArchComps.getFirstComp();
        If (comp.isArch())
          Archs.add(comp);
        While (comp != NULL)
          If  (!comp.findWithName
              (QueryComps[i].name))
            comp.setFlag(EQUAL_FAIL);
          comp := ArchComps.getNextComp();
          If (comp.isArch())
            Archs.add(comp);
        EndWhile
      EndFor
      For (i = 0; i < QueryConns.getSize(); i++)
        Conns conn = ArchConns.getFirstConn();
        While (conn != NULL)
          If (((!conn.getSrc() ==
              QueryConns[i].getSrc())) |
              (!(conn.getDst() ==
                  QueryConns[i].getDst())))
            conn.setFlag(EQUAL_FAIL);
          conn := ArchConns.getNextConn();
        EndWhile
      EndFor
      a := Archs.getNextArch();
    EndWhile
    integArch := RA.getNextArch();
EndWhile
End Difference Detection
```

기존의 아키텍처와 개발자의 새로운 아키텍처간 차이점을 탐색하는 과정은 두 단계로 진행된다. 먼저 기존 아키텍처가 포함하고 있는 각각의 아키텍처들을 리스트 형태로 관리하고, 리스트에 포함된 각각의 아키텍처와 새롭게 정의된 아키텍처의 컴포넌트와 커넥터에 대해 각각의 명세 정보를 비교하여 서로 다르게 정의된 부분을 탐지한다. 컴포넌트에 대한 명세 정보 비교는 컴포넌트 이름을 사용하여 비교하고, 커넥터에 대한 명세 정보 비교는 커넥터의 소스와 목표 컴포넌트 정보를 사용한다. 그리고 아키텍처가 하위 아키텍처를 포함하고 있을 경우는 아키텍처 리스트에 하위 아키텍처를 추가함으로써 하위 아키텍처에 대해서도 차이점 탐색 과정이 재귀적으로 수행될 수 있도록 한다.

<알고리즘 2>는 아키텍처간 차이점 정보를 기반으로 사이클 발생 부분을 다른 계층으로 분리함으로써 통합된 구조의 소프트웨어 아키텍처를 재구성하는 알고리즘이다.

<알고리즘 2> 아키텍처 통합 알고리즘

<Algorithm 2> Architecture integration algorithm

```
IntegrateCycledArch(2DGraphedArch1,
    2DGraphedArch2)
Begin Proc
  Search All Cycle(2DGraphedArch1,
      2DGraphedArch2)
  For each non-Cyclic Comps
    With Common Comp pair
      Merge Connector Relation Comp List
      Merge Include Relation Comp List
      Merge Super Relation Comp List
      Combine Designing Information
      Merge Description List
      Remain one of two and Remove the
other
    End With
  End For
  If not exist Cycle
    Return
End If
  For each Cycle
    newPlane1 = Make new plane for
     2DGraphedArch.Cycled Arch
    Move all 2DGraphedArch1.Cycled Comps to
     newPlane1
    newPlane2 = Make new plane for
     2DGraphedArch.Cycled Arch
    Move all 2DGraphedArch2.Cycled Comps to
     newPlane2
    Make new Include Relation Link
        from CurrentPlane.lastComp to
          newPlane1.HeadComp
    Make new Include Relation Link
        from CurrentPlane.lastComp to
          newPlane2.HeadComp
  End For
End Proc
```

통합 알고리즘은 사이클이 발생되지 않은 요소들의 정보를 최상위 계층의 아키텍처 내로 병합하고, 사이클이 발생된 요소들을 하위 아키텍처로 정의하여 다른 계층으로 분리시킨다. 그런 다음 계층간 포

함 관계와 연관 관계에 대한 정보를 생성하여 새로운 아키텍처를 기존의 아키텍처와 통합하여 하나의 소프트웨어 아키텍처를 재구성하게 된다.

4. 결론

이 연구에서는 기존의 소프트웨어 아키텍처와 이를 재사용하여 설계한 새로운 소프트웨어 아키텍처를 통합된 구조로 재구성할 수 있는 방법을 제안하였다. 그리고 통합된 구조의 소프트웨어 아키텍처를 명세하기 위한 XML 기반 소프트웨어 아키텍처 명세 구조를 정의하였다.

제안된 소프트웨어 아키텍처 재구성 방법은 다양한 관점에서 정의된 아키텍처들을 통합된 구조로 관리함으로써, 설계 과정에서 다양한 참조를 제공하였다. 통합된 구조의 관리 방식은 저장소에서의 아키텍처 관리를 위한 기본적인 분류를 제공함으로써 재사용성 향상을 위한 기반을 제공한다.

XML기반의 아키텍처 명세 구조를 사용하여 통합된 구조의 아키텍처를 관리함으로써 설계 단계에서의 구조적인 재사용이 가능하게 되었다. 그리고, 재사용 과정에서 발생하는 아키텍처의 변경 내용을 명세하기 위해 보다 유연성 있고 효율적인 방법을 제공할 수 있다.

향후 연구 과제로는 제안된 통합된 구조의 소프트웨어 아키텍처를 효율적으로 검색할 수 있는 검색 방법과 각각의 아키텍처를 버전별로 분류하여 관리할 수 있는 방법에 대한 연구가 필요하다.

※ 참고문헌

[1] R. Kazman, G. Abown, L. Bass and P. Clements, "Scenario-Based Analysis of Software Architecture," *IEEE Software*, PP.47-55, Nov., 1996.

[2] M. Shaw, "Architectural Issues in Software Reuse: It's Not Just the Functionality, It's the Packaging," *IEEE Symposium on Software Reuse*, IEEE Press, New York, 1995.

[3] R. Ramakrishnan and D. Janaki Ram, "Modeling Design Versions," *Proceedings of the 22nd International Conference on Very Large Databases '96*, pp.556-566, Sept., 1996.

[4] N. Medvidovic and R. N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages," *IEEE Transactions on Software Engineering*, Vol.26, No.1, Jan., 2000

[5] R. J. Allen, "A Formal Approach to Software Architecture," Ph.D. Thesis, Carnegie Mellon University, 1997.

[6] N. Medvidovic, D. S. Rosenblum and R. N. Taylor, "A Language and Environment for Architecture-Based Software Development and Evolution," *Proceedings of the 21st International Conference on Software Engineering*, May, 1999.

[7] Shawn Butler, David Diskin, Norman Howes and Kathleen Jordan, "Architectural Design of a Common Operating Environment," *IEEE Software*, pp.57-65, Nov., 1996.

[8] R. Conradi and B. Westfechtel, "Version Models for Software Configuration Management," *ACM Computing Surveys*, Vol.30, No.2, pp.232-282, Jun., 1998.

박 근 순

1962. 2 서울대학교
문리과대학 수학과(이학사)
1987.12 연세대학교
경영대학원 최고경영자과정
1995. 8 연세대학교
산업대학원 전자계산전공
(공학석사)
1967. 9 - 1977. 1 기업은행
(조사부, 전산부) 근무
제 3 차 경제개발추진계획
중소기업부문 수립
초기 Computer System
선정, 도입
1977. 1 - 1979. 2
서진전자악기공업주식회사
이사
1979. 3 - 1993.10 대우중공업
주식회사 정보관리담당 이사
1994.10 - 1996.12
고등기술연구원 연구위원
1992. 9 - 현 재
인하공업전문대학
컴퓨터정보공학부 교수
1995. 1 - 현 재
사단법인 법과 기술 연구소
이사
관심분야 : CASE tool,
ERP/CRM