論文
3-3-8

재사용을 위한 소프트웨어 아키텍쳐 명세와 재구성 방법 (Software Architecture Specification and Restructuring Method for Reuse)

이 윤 수*
(Yoon-Soo Lee)

요 약

소프트웨어 아키텍쳐는 설계 과정 동안 수정, 갱신 그리고 대체로 인해 재구성될 수 있으며, 소프트웨어 개발에 적합한 설계는 하나 이상 존재할 수 있기 때문에 다양한 관점에서 설계된 아키텍쳐를 재구성할 수 있는 방법과 아키텍쳐의 변경 요소들을 효율적으로 명세할 수 있는 방법이 필요하다.

이 논문에서는 기존에 이미 정의되어 있는 아키텍쳐와 이를 재사용하여 설계한 새로운 아키텍쳐를 통합된 구조의 소프트웨어 아키텍쳐로 재구성할 수 있는 방법을 제안하였다. 그리고, 통합된 구조의 소프트웨어 아키텍쳐 명세에 필요한 명세 요소와 명세 구조를 제안한다. 제안한 통합된 구조의 명세 방법은 아키텍쳐 설계 과정에서 다양한 개발자 관점에서 정의된 아키텍쳐들을 참조 및 재사용할 수 있는 방법을 제공하고, 아키텍쳐의 수정 및 갱신 그리고 대체와 같은 변경 요소에 대해 유연성 있는 명세 방법을 제공한다.

ABSTRACT

Software architectures can be restructured by modification and update during design processes, and appropriate software architectures for developments can be designed more than one. Therefore, it is required to specify efficiently the modification elements of architectures and restructuring method of designed architecture through various aspects.

In this paper, we proposedsoftware architecture restructuring mechanism that can restructure legacy architecture and a new software architecture designed with reuse of it in integrated form. And, we 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.

* 정회원 : 안산공과대학 컴퓨터정보과 조교수

논문접수 : 2002. 2. 20.
심사완료 : 2002. 3. 9.

1. 서론

소프트웨어 재사용은 소프트웨어 위기 현상을 극복하기 위한 방안으로 널리 연구되고 있는 소프트웨어 공학의 여러 분야 중에서 관심이 상대적으로 증가하고 있는 분야이다. 소프트웨어 재사용은 개발되어 있는 컴포넌트들을 새로운 소프트웨어 개발에 사용함으로써 소프트웨어 개발 기간을 단축시키고, 개발 비용의 절감과 소프트웨어의 생산성을 향상시킬 수 있는 효과를 얻을 수 있다[1].

전체적인 시스템을 컴포넌트의 조합으로 구성하는 소프트웨어 아키텍쳐는 설계 단계에서 전체적인 시스템의 문제점 파악과 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 소프트웨어들 간의 일반성을 향상시킬 수 있는 방법을 제공한다[2, 3].

소프트웨어 아키텍쳐는 전체 시스템의 구성을 개발 초기 단계에서 기술함으로써 설계 단계의 결정을 위한 지침을 제공하고, 개발자들로 하여금 시스템을 이해할 수 있게 하고, 시스템 구성에 대한 분석을 가능하게 한다. 또한, 설계 과정 동안 수정되거나 갱신, 대체로 인해 변경될 수 있다. 따라서, 개발자는 다양한 관점에서 설계된 아키텍쳐들의 서로 다른 구조들을 재구성할 수 있는 방법과 아키텍쳐의 변경 요소들을 효율적으로 명세할 수 있는 방법이 고려되어야 한다.

기존의 다양한 아키텍쳐 명세 언어들은 아키텍쳐 명세를 위한 정형화된 명세 요소들과 분석 방법을 제공함으로써 시스템의 전체적인 구조를 명세할 수 있는 방법을 제공한다[5]. 그러나, 정형화된 명세 언어들은 아키텍쳐 명세 구조 자체가 복잡하고, 수정과 대체로 인한 아키텍쳐의 변화에 유연성 있는 명세 방법을 제공하기가 어려웠다.

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

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

또한, 제안된 명세 구조는 XML형태로 구성함으로써 XML의 특징 중 하나인 구조적인 형태의 명세 구조를 나타낼 수 있게 하였다. 이는 개발자로 하여금 명세에 대한 특정 지식 없이 원하는 요구 사항 정보를 기술할 수 있고, 기존의 아키텍쳐 명세 언어의 명세 요소들을 대부분 지원하면서 아키텍쳐의 재사용과 수정, 대체 그리고 갱신과 같은 아키텍쳐의 변경 요소에 대해 유연성 있는 명세 방법을 제공한다.

2. 관련 연구

2.1 소프트웨어 아키텍쳐

최근 소프트웨어 공학에서의 연구 경향은 모듈 인터페이스 언어, 특정 영역 아키텍쳐, 구조적 기술 언어, 설계 패턴, 구조적 설계를 위한 기초, 그리고 구조적 설계 환경 영역에 대해 활발한 연구가 이루어지고 있다[4]. 소프트웨어 아키텍쳐는 프로그램과 시스템 컴포넌트, 그들의 상호관계에 대한 구조, 그리고 컴포넌트를 설계하고, 개발하기 위한 규칙 및 지침서라고 정의한다[10, 13].

소프트웨어 아키텍쳐에 관련된 최근 두 가지 경향은 복잡한 소프트웨어 시스템을 구조화하기 위해 메소드, 기술, 패턴 그리고 특징을 공유하는 것과 사용 가능한 프레임워크를 제공하기 위한 특정 영역의 이용 문제에 관한 것이다[13].

소프트웨어 개발에서 정형화된 소프트웨어 구조를 사용하게 되면 시스템 설계에서 추상화의 고수준으로 시스템을 표현함으로써 복잡한 시스템에 대한 이해를 돕고, 컴포넌트 라이브러리를 통해 다중 레벨에서 재사용이 가능하다. 그리고 정형화된 소프트웨어 구조 사용으로 인해 시스템의 발전 방향 예측이 가능하고, 소프트웨어 유지 보수자로 하여금 변화를

보다 쉽게 이해할 수 있도록 함으로 수정에 대한 비용을 비교적 정확하게 예측할 수 있게 한다. 또한, 컴포넌트간의 상호작용을 위한 연결을 조절함으로써 시스템의 성능과 상호작용 능력을 향상시킬 수 있으며, 시스템의 일관성 검사, 구조적 스타일의 일치, 특정 영역 분석을 위한 고수준의 형식을 제공할 수 있다.

2.2 소프트웨어 아키텍처 명세를 위한 명세 요소

소프트웨어 아키텍처에 대한 연구는 새로운 어플리케이션 개발 비용을 감소하고, 서로 밀접하게 관련된 다양한 소프트웨어들 간의 일반성을 향상시킬 수 있는 방향으로 진행되어 왔다. 일반적인 소프트웨어 아키텍처 특성에 기반한 새로운 소프트웨어 개발은 아키텍처를 구성하는 컴포넌트와 커넥터들의 전체적인 연결 구조과 통신 구조를 중심으로 아키텍처를 명세할 수 있는 방법을 필요로 한다[6].

아키텍처 명세를 위해서는 정형화된 명세 요소가 필요하고, 이러한 요소들을 분석하여 소프트웨어 아키텍처를 명세할 수 있어야 한다. 소프트웨어 아키텍처 명세 언어는 명세를 위한 정형화된 방법을 제공하고 있고, 응용의 구현보다는 고 수준에서의 전체적인 응용 구조를 정의하는데 중점을 두고 있다[7].

소프트웨어 아키텍처를 명세하기 위한 기존의 명세언어로 ACME, Aesop, C2, Rapide, SADL, UniCon, Weaves 등과 같은 많은 아키텍처 명세 언어가 이미 제안되어 사용되고 있다. 이러한 아키텍처 명세 언어들은 아키텍처를 구성하는 컴포넌트와 커넥터 단위에서의 명세 요소들을 정의하고 있다[5]. 컴포넌트와 커넥터에 대한 명세 요소로는 인터페이스, 타입, 의미 정보, 제약조건, 비함수적 속성 등이 있는데, 소프트웨어 아키텍처를 명세하기 위해서는 아키텍처를 구성하고 있는 컴포넌트와 커넥터 단위의 명세 요소들을 고려하여야 한다[5, 8].

2.3 XML 기반 소프트웨어 아키텍처 명세

소프트웨어 아키텍처는 전체 시스템을 컴포넌트간 상호작용과 연결을 관리하는 커넥터를 중심으로 분석하고, 기술함으로써 시스템의 전체적인 구조를 명세한다[6, 9]. 기존의 XML 기반 소프트웨어 아키텍처 명세[11, 13]는 소프트웨어 아키텍처도 하나의 컴포넌트라는 관점으로 명세에 접근하며, 컴포넌트들 간의 관련성을 컴포넌트들 간에 주고 받는 메시지로 정의하는 XML 형태의 소프트웨어 아키텍처 명세이다.

XML 기반 아키텍처 명세에서 아키텍처는 복합 컴포넌트와 동일하게 정의되며, 다수의 컴포넌트와 커넥터로 구성된다. 따라서, 아키텍처 재사용성을 고려할 때 소프트웨어 아키텍처 명세를 위해서는 컴포넌트 명세와 컴포넌트들 사이의 관련성 표현을 위한 커넥터의 명세가 필요하다[6, 11, 14].

아키텍처 기반 컴포넌트 명세에서는 아키텍처를 전체 시스템의 컴포넌트들 간의 상호작용과 연결을 관할하는 커넥터 중심으로 분석하고, 기술함으로써 시스템의 구조를 명세하였다. 컴포넌트와 컴포넌트 간의 관련성을 컴포넌트 간에 주고받는 메시지로 정의하여 XML형태의 소프트웨어 아키텍처 명세 구조를 정의하였으며, 재사용을 고려한 명세를 위해서 컴포넌트 명세와 컴포넌트 간의 관련성을 표현하는 커넥터의 명세로 정의하였다[11, 14].

기존의 아키텍처 명세는 컴포넌트와 커넥터에 대한 명세를 포함하고 있는데, 아키텍처를 구성하는 컴포넌트의 인터페이스와 메시지에 대해 명세함으로써 컴포넌트간 송수신되는 메시지의 목록을 정의하고 있다. 커넥터 명세는 반환 타입과 입출력 파라미터의 이름과 방향성 등과 같은 시그니쳐 정보에 대한 명세를 포함하고 있으며, 커넥터에 의한 메시지 전달에서 메시지를 발생시킨 컴포넌트와 메시지를 받아 처리하는 컴포넌트에 대한 명세를 포함하고 있다.

이 논문에서는 기존 아키텍처 명세의 이러한 구조적 특징을 이용하여 통합된 구조의 아키텍처 명세에 필요한 명세 요소와 명세 구조를 확장하여 정의한다.

3. 소프트웨어 아키텍처 명세와 재구성 방법

3.1 통합된 구조의 소프트웨어 아키텍처 명세

기존의 소프트웨어 아키텍처에 대한 재사용을 위해 개발자는 기존 아키텍처에 특정한 구조를 추가,

삭제 및 대체함으로써 새로운 아키텍쳐를 구성할 수 있다. 이 논문에서는 소프트웨어 아키텍쳐 설계 과정에서 재사용되는 아키텍쳐의 구성을 그대로 유지한 상태에서 새로 설계된 아키텍쳐의 구성을 통합된 구조로 재구성할 수 있는 방법을 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, 재구성된 소프트웨어 아키텍쳐가 포함하고 있는 각각의 아키텍쳐에 대한 전체적인 정보가 아키텍쳐 명세에 정의되어야 하고, 포함된 각각의 아키텍쳐를 표현하기 위한 정보들이 추가적으로 정의되어야 한다.

소프트웨어 아키텍쳐 재구성을 통한 통합과 이러한 통합된 소프트웨어 아키텍쳐에 포함된 각각의 아키텍쳐 표현을 위해서는 적합한 명세 요소에 대한 정의가 요구되는데, 이를 <표 1>을 통해 나타내었다.

확장된 아키텍쳐 명세는 통합된 구조의 소프트웨어 아키텍쳐가 포함하고 있는 각각의 아키텍쳐에 대한 정보를 표현하는 ArchInformation 태그와 서로 다른 개발자에 의해 설계된 각각의 아키텍쳐를 표현하기 위해 커넥터와 컴포넌트, 하위 아키텍쳐에 eachArchInfo 태그를 추가하였다. 또한, ArchInformation 태그에는 해당 아키텍쳐에 대한 아키텍쳐 이름과 개발자 정보를, eachArchInfo 태그에는 하위 아키텍쳐, 컴포넌트, 커넥터의 해당 아키텍쳐에 대한 정보를 이름 속성값으로 정의하였다.

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

<표 1> XML 기반 아키텍쳐 명세를 위한 명세 요소

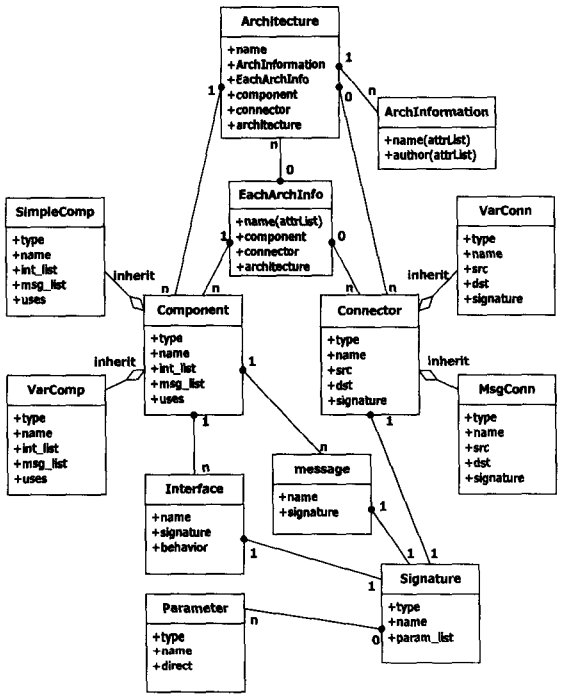
<Table 1> Specification Elements for Architecture Specification based XML

사용되는 태그	기존 XML기반 명세	통합을 위해 확장된 명세
Component	type, name, int_list, msg_list, uses	각각의 아키텍쳐 정보
Connector	type, name, src, dst,signature	각각의 아키텍쳐 정보
msg_list	number, message	추가 정보 없음
Interface	Name, signature	추가 정보 없음
Message	Name, signature	추가 정보 없음
Signature	name, type, param_list	추가 정보 없음
param_list	number, parameter	추가 정보 없음
Parameter	type, name, direct	추가 정보 없음
Architecture	name, component, connector, architecture	
ArchInformation	정보 없음	포함된 아키텍쳐 정보(아키텍쳐 이름과 개발자 정보)
EachArchInfo	정보 없음	각각의 아키텍쳐 정보(아키텍쳐, 컴포넌트, 커넥터들의 해당 아키텍쳐에 따른 정보)

<리스트 1> 확장된 XML 기반 아키텍쳐 명세의 DTD

```
<?xml version="1.0"?>
<!ELEMENT architecture ((eachArchInfo+, component+, connector*, architecture*)
                |(name, ArchInformation*, component+, connector*, architecture*))>
<!ELEMENT component ((eachArchInfo+)|(type?, name, int_list?, msg_list?))>
<!ELEMENT connector ((eachArchInfo+)|(type?, name, src, dst, signature?))>
<!ELEMENT eachArchInfo ((type?, name, src, dst, signature?)
                |(name, component+, connector*,architecture*)
                |(name, int_list?, msg_list?))>
<!ATTLIST ArchInformation name CDATA #REQUIRED>
<!ATTLIST ArchInformation author CDATA #REQUIRED>
<!ATTLIST eachArchInfo name CDATA #IMPLIED>
<!ELEMENT int_list (number, interface*)><!ELEMENT msg_list (number, message*)>
<!ELEMENT interface (name?, signature)><!ELEMENT message (name?, signature)>
<!ELEMENT signature (name?, type?, param_list)>
<!ELEMENT param_list (number, parameter*)>
<!ELEMENT parameter (type?, name?, direct?)><!ELEMENT name (#PCDATA)>
<!ELEMENT type (#PCDATA)><!ELEMENT uses (#PCDATA)>
<!ELEMENT direct (#PCDATA)><!ELEMENT number (#PCDATA)>
<!ELEMENT src (#PCDATA)><!ELEMENT dst (#PCDATA)>
<!ELEMENT behavior (name?, modifies?, ensures)>
<!ELEMENT modifies (#PCDATA)><!ELEMENT ensures (#PCDATA)>
```

정의한 DTD를 기반으로 하는 확장된 XML 기반 소프트웨어 아키텍쳐 명세 구조의 클래스 다이어그램은 [그림 1]과 같이 표현하였다.



[그림 1] 확장된 아키텍쳐 명세 구조 클래스 다이어그램

[Fig. 1] Extended Architecture Specification Structure Class Diagram

개발자들의 다양한 관점에서 설계된 아키텍쳐들을 모두 포함하는 통합된 구조의 소프트웨어 아키텍쳐 명세에서 아키텍쳐 이름과 개발자 이름과 같은 각각의 아키텍쳐 설계 정보를 표현하기 위한 ArchInformation 클래스가 아키텍쳐 명세의 최상위에 포함되어 있으며, 컴포넌트와 커넥터로 구성되는 아키텍쳐에는 eachArchInfo 클래스를 추가하여 개발자들의 서로 다른 관점에서의 아키텍쳐 정의를 모두 반영할 수 있도록 정의하고 있다. eachArchInfo 클래스는 컴포넌트 형태의 하부 아키텍쳐를 포함할 수 있으므로 Architecture 클래스의 컴포넌트와 커넥터 클래스를 포함하고 있는 형태를 가진다. 아키텍쳐를 구성하고 있는 각각의 컴포넌트와 커넥터에 대한 명세 클래스는 기존의 XML 명세 구조와 동일한 구조를 사용하고 있다.

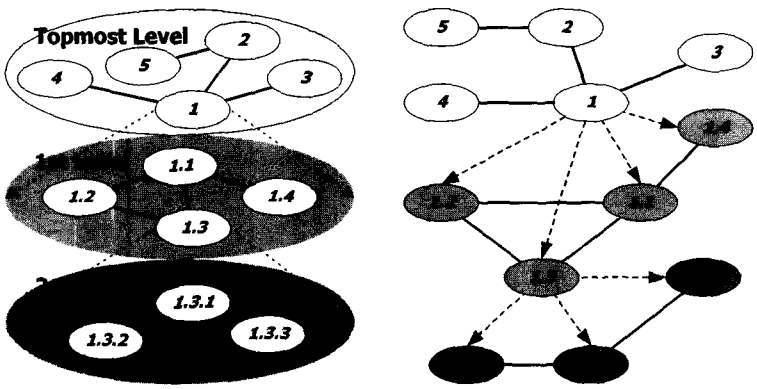
3.2 소프트웨어 아키텍처 재구성

소프트웨어 아키텍처 설계 과정에서 개발자는 기존에 이미 정의되어 있는 아키텍처들을 재사용하여 자신의 개발 환경에 적합한 새로운 소프트웨어 아키텍처를 설계할 수 있다. 이 논문에서는 이러한 설계 과정에서 새로 구성된 아키텍처를 기존의 소프트웨어 아키텍처 설계를 그대로 유지하는 상태에서 저장소에 반영하기 위해 통합된 구조의 재구성 소프트웨어 아키텍처 명세 구조를 정의한다.

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

이러한 아키텍처들간 서로 다르게 정의된 부분에 대해 추출한 정보를 기반으로 기존의 정의된 아키텍처와 아키텍처 설계 과정에서 개발자가 새로 설계한 아키텍처 사이의 공통된 관점과 서로 다른 관점으로 정의된 부분을 구분하여 아키텍처 명세에 적용함으로써 하나의 통합된 구조의 소프트웨어 아키텍처를 명세할 수 있는 방법을 제공하고 있다. [그림 2]는 이 논문에서 제안하는 소프트웨어 아키텍처 재구성 과정의 이해를 위해 계층적인 아키텍처 구조와 아키텍처의 통합과 명세를 위해 아키텍처를 커넥터의 연관 관계와 계층간 포함 관계를 사용하여 이차원 평면 구조로 나타낸 것이다.

[그림 2]에서 실선으로 표현된 부분은 컴포넌트간 교환되는 메시지를 나타내는 커넥터를 의미하고, 점선으로 표현된 부분은 상위 아키텍처를 구성하는 컴포넌트가 하위 계층의 아키텍처를 포함하고 있음을 나타낸다. 이는 커넥터에 의해 표현되는 컴포넌트간 관련성과 계층간 포함 관계를 모두 표현하고 있다.



[그림 2] 개념적인 소프트웨어 아키텍처 구조

[Fig. 2] Conceptual Software Architecture Structure

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 논문에서는 컴포넌트와 하위 계층 아키텍쳐와 계층간의 포함 관계가 뒤바뀐 형태를 아키텍쳐의 사이클이라 정의하고, 이러한 사이클이 발생한 경우 사이클 발생 부분을 모두 명세에 반영하는 통합된 형태의 소프트웨어 아키텍쳐를 구성한다.

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 논문에서 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.

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 아키텍쳐간 차이점 정보를 기반으로 사이클 발생 부분을 다른 계층으로 분리함으로써 통합된 구조의 소프트웨어 아키텍쳐를 재구성하는 알고리즘이다.

<알고리즘 1> 통합된 구조의 소프트웨어 아키텍쳐 재구성 알고리즘

```
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 아키텍쳐 내로 병합하고, 사이클이 발생된 요소들을 하위 아키텍쳐로 정의하여 다른 계층으로 분리시킨다. 그런 다음 계층간 포함 관계와 연관 관계에 대한 정보를 생성하여 새로

운 아키텍처를 기존의 아키텍처와 통합하여 하나의 소프트웨어 아키텍처를 재구성하게 된다.

4. 실험 및 평가

이 실험에서는 제안한 통합된 구조의 소프트웨어 아키텍처 재구성과 명세를 위한 모듈들의 기능성을 평가하기 위해 다양한 형태의 소프트웨어 아키텍처를 정의하고, 정의한 소프트웨어 아키텍처들과 유사한 형태의 소프트웨어 아키텍처를 정의하여 저장소에 저장하였다. 통합된 구조의 소프트웨어 아키텍처는 하나의 아키텍처가 3개에서 7개의 내부 아키텍처를 포함하도록 정의하였다.

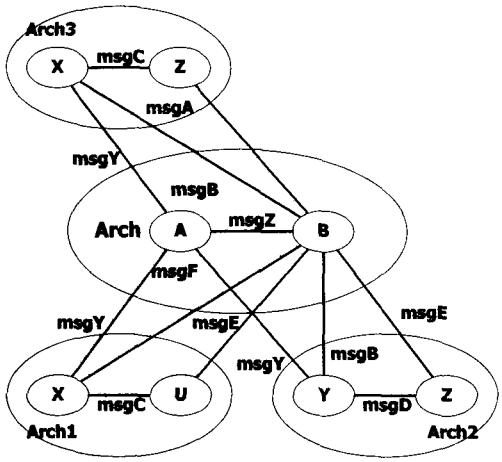
통합된 구조의 소프트웨어 아키텍처 재구성을 위한 실험에서 개발자는 기존의 아키텍처를 재사용하여 새로운 아키텍처를 설계하여야 한다. 개발자는 기존 저장소 아키텍처들의 명세 내용을 확인하여 재사용에 적합한 아키텍처를 선택하고, 선택한 아키텍처를 재사용하여 자신의 개발 환경에 적합한 형태로 수정하여 새로운 아키텍처를 설계한다. 최종적으로 설계된 새로운 소프트웨어 아키텍처는 기존의 아키텍처와 통합된 구조로 재구성된다.

실험에서 우선 개발자는 인터페이스를 통해 저장소에 저장되어 있는 기존의 아키텍처 명세를 참조하고, 자신의 개발 환경에 적합한 아키텍처를 선택하게 된다. 선택한 아키텍처를 재사용하여 개발자는 아키텍처 생성 인터페이스를 통해 새로운 아키텍처를 설계하게 된다.

[그림 3]은 개발자가 선택한 저장소의 소프트웨어 아키텍처의 명세 구조를 보여주고 있다.

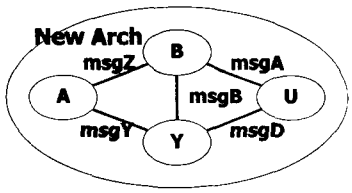
개발자는 기존의 재구성 소프트웨어 아키텍처 명세가 포함하고 있는 각각의 아키텍처를 참조하고, 이를 재사용하여 자신의 개발 환경에 적합한 새로운 소프트웨어 아키텍처를 설계하게 된다.

[그림 4]는 개발자가 기존 아키텍처를 재사용하여 새로 정의한 아키텍처 명세 구조를 보여주고 있다.



[그림 3] 재사용을 위한 기존 소프트웨어 아키텍처 명세 구조

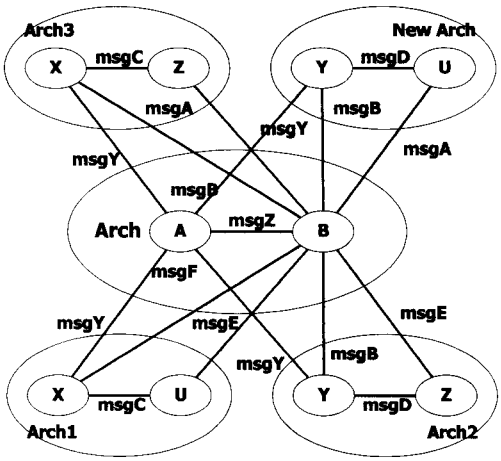
[Fig. 3] Legacy Software Architecture Specification Structure for Reuse



[그림 4] 새로 정의한 소프트웨어 아키텍처 명세 구조

[Fig. 4] New Defined Software Architecture Specification Structure

설계 과정에서 새로 정의된 소프트웨어 아키텍처는 관리를 위해 기존 아키텍처와 통합된 구조로 재구성된다. 우선 차이점 탐색 모듈에 의해 기존 아키텍처와의 사이클 발생 부분에 대한 정보를 생성하게 되고, 이러한 사이클 발생 정보를 기반으로 통합 모듈은 커넥터에 의한 연관 관계와 계층간 포함 관계 정보를 생성하여 통합된 구조의 아키텍처를 구성하게 된다.



[그림 5] 통합된 형태의 소프트웨어 아키텍쳐 명세 구조
[Fig. 5] Software Architecture Specification Structure of Integrated Form

[그림 5]는 검색 모듈에 의해 검색된 기존의 아키텍쳐와 개발자의 새로운 아키텍쳐가 통합된 구조로 재구성된 아키텍쳐 구조를 나타내고 있고, <리스트 2>는 통합된 구조로 재구성된 소프트웨어 아키텍쳐의 연관 관계와 포함 관계 정보를 기반으로 XML 형태의 명세 구조로 표현된 아키텍쳐 명세의 일부를 나타내고 있다.

<리스트 2> 통합된 재구성 소프트웨어 아키텍쳐 명세의 일부

```
<!DOCTYPE recon SYSTEM "ArchSpec.dtd">
<architecture><name>Arch</name>
<ArchInformation name="Arch1" number="1"author="AuthorA"date="2000/09/17"/>
<ArchInformation name="Arch2" number="1"author="AuthorB"date="2000/09/19"/>
<ArchInformation name="Arch3" number="1"author="AuthorC"date="2000/09/21"/>
<ArchInformation name="New Arch" number="1"author="AuthorD"date
="2000/09/25"/>
  <component><eachArchInfo><name>A</name></eachArchInfo></component>
  <component><eachArchInfo><name>B</name></eachArchInfo></component>
  <component><eachArchInfo name="Arch1"><name>X</name></eachArchInfo>
    <eachArchInfo name="Arch2"><name>Y</name></eachArchInfo>
    <eachArchInfo name="Arch3"><name>X</name></eachArchInfo>
    <eachArchInfo name="New Arch"><name>Y</name></eachArchInfo>
</component>
.....
  <connector><eachArchInfo><name>msgZ</name>
    <src>A</src><dst>B</dst></eachArchInfo></connector>
  <connector><eachArchInfo name="Arch1">
    <name>msgY</name><src>A</src><dst>X</dst></eachArchInfo>
  <eachArchInfo name="Arch2">
    <name>msgY</name><src>A</src><dst>Y</dst></eachArchInfo>
  <eachArchInfo name="Arch3">
    <name>msgY</name><src>A</src><dst>X</dst></eachArchInfo>
  <eachArchInfo name="New Arch">
    <name>msgY</name><src>A</src><dst>Y</dst></eachArchInfo>
.....
</architecture>
```

실험을 통해 제안한 소프트웨어 아키텍쳐 재구성 방법은 기존에 이미 정의되어 있는 소프트웨어 아키텍쳐와 개발자가 이를 재사용하여 새로 설계한 소프트웨어 아키텍쳐가 하나의 통합된 구조로 재구성할 수 있음을 검증하였고, 통합된 구조로 재구성된 소프트웨어 아키텍쳐가 기존의 XML 기반 아키텍쳐 명세 구조를 확장하여 정의한 XML 기반 아키텍쳐 명세 구조로 명세됨을 확인하였다.

5. 결론

이 논문에서는 기존의 소프트웨어 아키텍쳐와 이를 재사용하여 설계한 새로운 소프트웨어 아키텍쳐를 통합된 구조로 재구성할 수 있는 방법을 제안하였다. 그리고, 통합된 구조의 소프트웨어 아키텍쳐를 명세하기 위한 XML 기반 소프트웨어 아키텍쳐 명세 구조를 정의하였다.

제안된 소프트웨어 아키텍쳐 재구성 방법은 다양한 관점에서 정의된 아키텍쳐들을 통합된 구조로 관리함으로써 설계 과정에서 다양한 참조를 제공하였다. 통합된 구조의 관리 방식은 저장소에서의 아키텍쳐 관리를 위한 기본적인 분류를 제공함으로써 재사용성 향상을 위한 기반을 제공한다. 그리고, 인터페이스를 통해 설계된 새로운 아키텍쳐는 각각의 재구성 모듈을 통해 기존 아키텍쳐와 통합되어 XML 명세 구조로 명세됨을 확인할 수 있었다. 또한, XML기반의 아키텍쳐 명세 구조를 사용하여 통합된 구조의 아키텍쳐를 관리함으로써 설계 단계에서의 구조적인 재사용이 가능하게 되었다.

향후 연구 과제로는 제안한 통합된 구조의 소프트웨어 아키텍쳐를 효율적으로 검색할 수 있는 검색 방법과 각각의 아키텍쳐를 버전별로 분류하여 관리할 수 있는 버전 관리에 대한 연구가 필요하다.

※ 참고 문헌

[1] C. W. Krueger, "Software Reuse," ACM Computing Surveys, Vol.24, No.2, 1992.

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

[3] 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.

[4] D. Garlan and M. Shaw, "An Introduction to Software Architecture," Advances in Software Engineering and Knowledge Engineering, Volum2, World Scientific Publishing, Singarpore, 1993.

[5] 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

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

[7] 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.

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

[9] D. Garlan and R. Allen, "A Formal Basis for Architectural Connection," *ACM Transactions on Software Engineering and Methodology*, Vol.6, No.3, Jul., 1997.

[10] 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.

[11] Y. S. Lee, K. S. Yoon and C. J. Wang, "Component Retrieval Based on Architecture for Reuse," *Proceedings on 16th International Federation for Information Processing*, Aug., 2000.

[12] C. D. Ahn, S. G. Lee, C. J. Wang, "Restructuring Model for Collaborative Multimedia Authoring," *Proceedings on Internet and Multimedia Systems and Applications*, Nov., 2000.

[13] D.E. Perry and D. Garlan, "Introduction to the Spectial Issue on Software Architecture," IEEE Transaction on Software Engineering, vol. 21, No. 4, Apr. 1995.

[14] 이윤수, 윤경섭, 왕창종, "재사용을 위한 XML 기반 소프트웨어 아키텍쳐 명세 언어," *한국정보처리학회, 정보처리논문지*, 제7권, 제3호, 2000년, 3월.

이 윤 수

1988년 인하대학교 전자계산학과 졸업(학사)

1990년 인하대학교 대학원 전자계산학과 졸업(이학석사)

2000년 인하대학교 대학원 전자계산공학과 졸업(공학박사)

1995년 ~ 1996년 인하대학교 전자계산공학과 전임대우강사

1998년 ~ 현재 안산공과대학 전자계산소장

1996년 ~ 현재 안산공과대학 컴퓨터정보과 조교수

관심분야 : 컴포넌트 기반 소프트웨어 공학, 분산객체 컴퓨팅, 멀티미디어 시스템