# 재사용을 위한 소프트웨어 아키텍처 재구성

안 치 돈[†] · 왕 창 종[††]


## 요 약

소프트웨어 아키텍처는 설계 과정 동안 수정과 대체로 인해 변경될 수 있고, 소프트웨어 개발에 적합한 설계는 하나 이상 존재할 수 있으므로 개발자는 다양한 관점에서 설계된 아키텍처의 서로 다른 버전들을 유지 관리할 수 있는 방법과 아키텍처의 변경 요소들을 효율적으로 명세할 수 있는 방법이 필요하다. 본 논문에서는 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 통합 구조의 명세 방법은 아키텍처 설계 과정에서 다양한 개발자 관점에서 정의된 아키텍처들을 참조 및 재사용할 수 있는 방법을 제공한다.


**키워드 : 아키텍처 재구성, 명세, 재사용**

# Software Architecture Restructuring for Reuse

Chi-Don Ahn[†] · Chang-Jong Wang[††]


## ABSTRACT

Software architectures can be restructured by modification and replacement during design processes, and appropriate software architectures for developments can be more than one. Therefore, developers are required to specify efficiently the modification elements of architectures, and manage different versions of an architecture designed for various aspects. In this paper, we propose a mechanism that can restructure legacy architecture and a new software architecture designed with reuse of it in integrated form, and define the specification elements and structure of the proposed architecture restructuring specification. It provides the method that can reference and reuse architectures designed with various aspects of developers in architecture design processes.


**Key word : Architecture Restructuring, Specification Reuse, Design Versioning**

## 1. 서 론

소프트웨어 아키텍처는 새로운 응용 개발 비용을 감소하고, 서로 밀접하게 관련된 다양한 소프트웨어 제품군 간의 일반성을 향상시킬 수 있는 방법을 제공한다[1, 2]. 소프트웨어 아키텍처는 설계 과정 동안 수정과 대체로 인해 변경될 수 있고, 새로운 소프트웨어 개발에 적합한 설계는 하나 이상 존재할 수 있다. 따라서 개발자는 다양한 관점에서 설계된 아키텍처들의 서로 다른 버전들을 유지 관리할 수 있는 방법과 아키텍처의 변경 요소들을 효율적으로 명세할 수 있는 방법이 고려되어야 한다[3].

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

본 논문에서는 기존에 정의되어 있는 아키텍처와 이를 재사용하여 설계한 새로운 아키텍처를 통합된 구조의 아키텍처로 재구성할 수 있는 방법을 제안하고 있다. 그리고 통합된 구조의 아키텍처 명세에 필요한 명세 요소와 명세 구조에 대해 정의하고 있다. 제안하는 아키텍처 재구성 방법은 아키텍처 설계 과정에서 다양한 개발자 관점에서 정의된 아키텍처들을 참조 및 재사용할 수 있는 방법을 제공하고 있다. 그리고 서로 관련된 아키텍처들을 통합된 구조로 재구성하여 저장함으로써 아키텍처 명세 관리를 위한 기본적인 분류를 제공한다.

XML 기반 아키텍처 명세 구조는 기존의 아키텍처 명세 언어의 명세 요소들을 대부분 지원하면서 아키텍처의 재사용과 수정과 대체 등의 아키텍처의 변경 요소에 대해 유연성 있는 명세 방법을 제공한다. 그리고, 기존의 명세 언어와는 달리 비정형적인 방법에 의한 명세 방법을 제공하므로 보다 쉬운 형태의 아키텍처 명세가 가능하다.

## 2. 아키텍처 명세와 버전 모델

아키텍처에 대한 연구는 새로운 응용 개발 비용을 감소

† 정 회 원 : 인하대학교 대학원 전자계산공학과
†† 정 회 원 : 인하대학교 전자계산공학과 교수
논문접수 : 2000년 11월 20일, 심사완료 : 2001년 2월 5일

시키고[5, 6], 서로 밀접하게 관련된 다양한 소프트웨어 제품군 간의 일반성을 향상시킬 수 있는 방향으로 진행되어 왔다. 일반적인 아키텍쳐 특성에 기반한 새로운 소프트웨어 개발은 아키텍쳐를 구성하는 컴포넌트와 커넥터들의 전체적인 연결 구조와 통신 구조를 중심으로 아키텍쳐를 명세할 수 있는 방법을 필요로 한다[7-10].

### 2.1 아키텍쳐 명세를 위한 명세 요소

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[9, 11, 12].

아키텍쳐를 명세를 위해 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, 타입, 의미 정보(semantics), 제약조건, 비함수적 속성 등이 있는데, 소프트웨어 아키텍쳐를 명세하기 위해서는 아키텍쳐를 구성하고 있는 컴포넌트와 커넥터 단위의 이러한 명세 요소들을 고려하여야 한다[4, 14].

### 2.2 XML 기반 아키텍쳐 명세

아키텍쳐는 전체 시스템을 컴포넌트간 상호작용과 연결을 관리하는 커넥터를 중심으로 분석하고 기술함으로써 시스템의 전체적인 구조를 명세한다[7, 9]. 기존의 XML 기반 아키텍쳐 명세[10, 15, 16]는 소프트웨어 아키텍쳐도 하나의 컴포넌트라는 관점으로 명세에 접근하며, 컴포넌트들간 관련성을 컴포넌트간 주고 받는 메시지로 정의하는 XML 형태의 아키텍쳐 명세 모델이다. XML 기반 아키텍쳐 명세 모델에서 아키텍쳐는 복합 컴포넌트와 동일하게, 다수의 컴포넌트와 커넥터로 구성된다[7, 10, 15, 16].

아키텍쳐 기반 컴포넌트 명세에서는 아키텍쳐를 컴포넌트들 간의 상호 작용과 연결을 관할하는 커넥터 중심으로 분석하고, 기술함으로써 시스템의 구조를 명세하였다. 컴포넌트와 컴포넌트 간의 관련성을 컴포넌트 간에 주고받는 메시지로 정의하여 XML형태의 아키텍쳐 명세 모델을 정의하였으며, 재사용을 고려한 명세를 위해 컴포넌트와 커넥터의 명세로 나누어 정의하였다[10, 15, 16]. 본 논문에서는 기존 아키텍쳐 명세의 이러한 구조적 특징을 이용하여 통합된 구조의 아키텍쳐 명세에 필요한 명세 요소와 명세 구조를 확장하여 정의한다.

### 2.3 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 대해서 정의한다[3]. 버전 모델은 또한 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 버전 모델이 표현하는 상태나 기본 요소와 관련된 변화에 의해 특정지어 질 수 있는지 여부를 정의한다[17].

본 논문에서는 기존 아키텍쳐의 재사용 과정에서 아키텍쳐를 구성하는 컴포넌트와 커넥터를 대체하는 방식으로 새로운 아키텍쳐 설계가 가능하다는 점에서 대체값에 의한 버전화 방식을 도입하며, 변경 요소에 대한 명세는 특정 컴포넌트와 커넥터 명세에 대해 이루어질 수 있다는 점에서 원자 객체들을 사용한 버전 제어 방식과 구조적 명세의 모든 계층에 대한 버전화 방법을 도입한 새로운 형태의 명세 방식을 제안한다.

## 3. 아키텍쳐 재구성을 위한 명세

이 장에서는 아키텍쳐 설계 과정에서 재사용되는 아키텍쳐의 구성을 그대로 유지한 상태에서 새로 설계된 아키텍쳐의 구성을 통합된 구조로 재구성할 수 있는 메커니즘을 제안한다.

### 3.1 통합된 구조의 아키텍쳐 명세

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

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

〈표 1〉 XML 기반 아키텍처 명세를 위한 명세 요소

| 태 그 | 기존 명세 | 확장된 명세 |
|---|---|---|
| Component | type, name, int_list, msg_list, uses | |
| Connector | type, name, src, dst, signature | |
| int_list | number, interface | 추가 정보 없음 |
| 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 | |
| | 정보 없음 | |
| | 정보 없음 | |

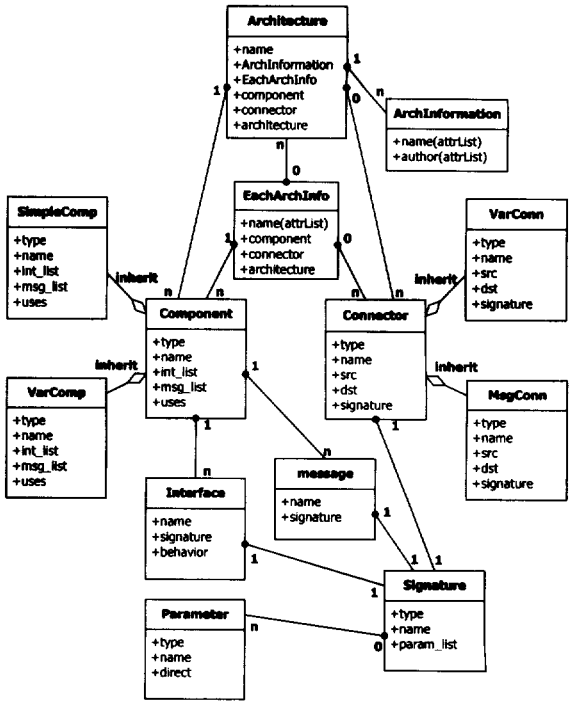
확장된 아키텍처 명세는 통합된 구조의 아키텍처가 포함하고 있는 각각의 아문에서 제안하는 통합된 구조의 재구성 소프트웨어 아키텍처를 XML 기반으로 명세하기 위한 DTD를 나타내고 있다.

〈리스트 1〉 확장된 XML 기반 아키텍처 명세의 DTD

```
<?xml version="1.0"?>
<!ELEMENT architecture ((eachArchInfo+, component+,
        connector*, architecture*) |(name, ArchInfo*,
        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 ArchInfo name CDATA #REQUIRED>
<!ATTLIST ArchInfo 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 아키텍처 명세 구조는 (그림 1)을 통해 클래스 다이어그램으로 표현하였다.

개발자들의 다양한 관점에서 설계된 아키텍처들을 모두 포함하는 통합된 구조의 소프트웨어 아키텍처 명세에서 아키텍처 이름과 개발자 이름과 같은 각각의 아키텍처 설계 정보를 표현하기 위한 ArchInfo 클래스가 아키텍처 명세의 최상위에 포함되어 있으며, 컴포넌트와 커넥터로 구성되는 아키텍처에는 eachArchInfo 클래스를 추가하여 개발자들의 서로 다른 관점에서의 아키텍처 정의를 모두 반영할 수 있도록 정의하고 있다.




(그림 1) 확장된 아키텍처 명세 클래스 다이어그램

또한 eachArchInfo 클래스는 컴포넌트 형태의 하부 아키텍처를 포함할 수 있으므로 Architecture 클래스의 컴포넌트와 커넥터 클래스를 포함하고 있는 형태를 가진다. 아키텍처를 구성하고 있는 각각의 컴포넌트와 커넥터에 대한 명세 클래스는 기존의 XML 명세 구조와 동일한 구조를 사용하고 있다.

### 3.2 아키텍처 재구성

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

기존에 정의되어 있는 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. 이는 통합된 구조의 아키텍처 명세에 있어 서로 다른 부분의 명세에만 각각의 아키텍처에 대한 정보를 추가하여 명세에 적용하기 위한 것이다.

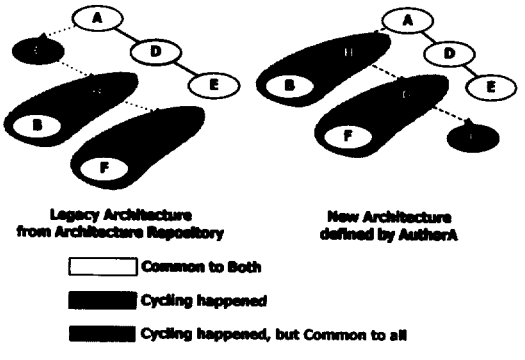
아키텍처들간에 서로 다르게 정의된 부분에 대해 추출한 정보를 기반으로 기존에 정의된 아키텍처와 설계 과정에서 개발자가 새로 설계한 아키텍처 사이의 공통된 관점과 서로 다른 관점으로 정의된 부분을 구분하여 아키텍처 명세에 적용함으로써 하나의 통합된 구조의 아키텍처를 명세할 수 있다. 계층적인 아키텍처 구조와 아키텍처의 통합 명세를 위해 아키텍처를 커넥터의 연관 관계와 계층간 포함 관계를 나타낼 수 있도록 컴포넌트간 관련성과 계층간 포함 관계를 모두 표현해야 한다.

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

본 논문에서는 컴포넌트와 하위 계층 아키텍처간, 그리고 계층간의 포함 관계가 뒤바뀐 형태를 아키텍처의 사이클이라 정의하고, 이러한 사이클이 발생한 경우 사이클 발생 부분을 모두 명세에 반영하는 통합된 형태의 소프트웨어를 구성한다.

(그림 2)는 저장소의 기존 아키텍처와 개발자에 의해 정의된 새로운 아키텍처간 사이클이 발생한 예를 보여주고 있다. 그림에서 흰색으로 표시된 컴포넌트들은 기존 아키텍처와 공통된 관점에서 설계된 포함 관계가 일치하는 형태의 컴포넌트들을 나타내며, 짙은 회색으로 표현된 컴포넌트들은 설계된 아키텍처 명세 사이에서 사이클이 발생한 경우를 표현한다.

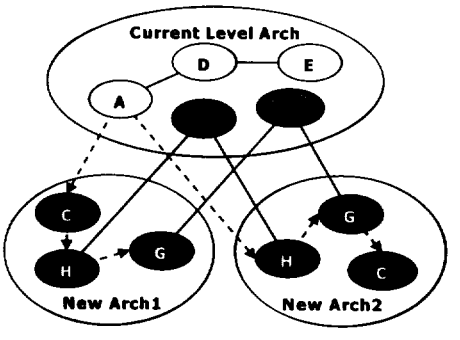
옅은 회색으로 표현된 컴포넌트들은 실제로 사이클이 발생한 부분에 포함되지만 컴포넌트간 관련성은 일치하는 부분을 나타낸다. 점선으로 표현한 화살표는 하위 계층으로의 커넥터를 나타내며, 실선은 같은 계층 내의 커넥터를 의미한다.




(그림 2) 아키텍처들간 사이클 발생 예

본 논문에서는 이러한 사이클이 발생했을때, 사이클이 발생된 컴포넌트들을 서로 다른 평면으로 분리하고, 공통된 관점에서 정의된 컴포넌트들에 대한 연관 관계를 생성함으로써 기존 아키텍처와 새로 설계된 아키텍처를 모두 반영하는 통합된 형태의 소프트웨어 아키텍처를 명세한다.

(그림 3)은 (그림 2)에서 사이클이 발생한 컴포넌트를 나타내는 C, G, H를 서로 다른 평면으로 분리하여 기존 아키텍처와 새로 설계된 아키텍처를 모두 반영하고 있는 통합된 구조의 소프트웨어 아키텍처를 재구성한 예를 보여주고 있다.




(그림 3) 통합된 구조의 소프트웨어 아키텍처

(그림 3)에서 Current Level Arch와 New Arch1을 통해 기존에 정의된 아키텍처 구조가 변형 없이 수용되었고, Current Level Arch와 New Arch2를 통해 개발자가 새로 설계한 아키텍처 구조 또한 변형 없이 수용되어 통합된 구조의 소프트웨어 아키텍처로 재구성 되었음을 알 수 있다.

기존의 아키텍처와 개발자가 새로 정의한 아키텍처를 통합된 구조의 소프트웨어 아키텍처로 재구성하여 관리하는 방법은 아키텍처 단위의 계층간 포함 관계에서 사이클 발생 부분을 발견하여 적용 가능할 뿐만 아니라, 컴포넌트 단위와 커넥터 단위의 연관 관계에서 사이클을 발견하여 통합된 구조의 소프트웨어 아키텍처

본 논문에서 제안하는 서로 다른 개발자에 의해 정의된 소프트웨어 아키텍처를 구조별로 따로 제공하는 것이 아니라 하나의 통합된 구조로 제공하는 방법은 개발자가 새로운 소프트웨어 개발을 위한 아키텍처 설계 단계에서 다양한 관점에서 정의된 소프트웨어 아키텍처를 참조하여 재사용할 수 있도록 함으로써 기존의 아키텍처에 대한 재사용성과 개발에서의 유연성을 향상시킬 수 있음을 의미한다.

### 3.3 소프트웨어 아키텍처 재구성 모듈 설계

이 절에서는 저장소에 저장되어 있는 기존의 아키텍처와

개발자가 아키텍처 설계 과정에서 이러한 저장소의 아키텍처를 재사용하여 새로 설계한 아키텍처를 하나의 통합된 구조로 명세하는데 필요한 모듈들을 설계한다. 통합된 구조의 아키텍처를 명세하기 위해서는 개발자의 새로운 아키텍처 설계를 XML 기반 아키텍처 명세 구조로 명세하는 모듈과 기존의 아키텍처 정의간 공통적인 관점에서 정의된 부분과 사이클 발생 부분에 대한 정보를 추출하는 모듈, 그리고 이러한 추출 정보를 기반으로 아키텍처들을 통합하는 모듈이 필요하다.

개발자가 기존의 소프트웨어 아키텍처를 재사용하여 새로 설계한 소프트웨어 아키텍처는 명세 모듈을 통해 XML 기반 아키텍처 명세 구조로 변환된다. XML 형태로 명세된 새로운 아키텍처는 기존의 아키텍처 구성을 유지한 상태에서 통합된 구조의 아키텍처로 재구성 되기 위해 아키텍처간 공통 부분과 사이클 발생 부분에 대한 정보를 추출하는 차이점 탐색 모듈로 입력된다. 차이점 탐색 모듈을 통해 추출된 아키텍처의 공통 부분과 사이클 발생 부분에 대한 정보를 기반으로 통합 모듈은 사이클이 발생한 아키텍처, 컴포넌트, 그리고 커넥터들을 다른 계층으로 분리시키고 서로간의 연관 관계와 포함 관계에 대한 정보를 생성함으로써 재구성된 소프트웨어 아키텍처가 하나의 통합된 구조로 명세된다.

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

```
method Difference Detection between Architectures
Query Architecture : Q
Retrieved Architectures : RA
Begin Find Difference Detection
 For each Retrieved Arch
  Make Arch List using Archs included in Retrieved Arch
  For each Arch in Arch List
   Make Comp List using Arch
   Make Comp List using Query Arch
   Make Conn List using Arch
   Make Conn List using Query Arch
   For each Comp in Arch
     If Comp is Arch Then add Comp to Archs
     If Query Comp's name not exists in Arch's Comp name
     Then Set flag to EQUAL_FAIL
   End For
   For each Conn in Arch
     If Conn's src and dest not exists in Arch's Conn Then Set
     flag to EQUAL_FAIL
   End For
   Get Next Arch in Retrieved Arch
  End For
  Get Next Integrated Arch in Repository
End For
End Difference Detection
```

〈알고리즘 1〉 아키텍처간 차이점 탐색 알고리즘

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 명칭과 방향성을 비교하기 위해 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.

```
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
 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
```

〈알고리즘 2〉 아키텍처 통합 알고리즘

통합 알고리즘은 사이클이 발생되지 않은 요소들의 정보를 최상위 계층의 아키텍처 내로 병합하고, 사이클이 발생된 요소들을 하위 아키텍처로 정의하여 다른 계층으로 분리시킨다. 그런 다음 계층간 포함 관계와 연관 관계에 대한 정보를 생성하여 새로운 아키텍처를 기존의 아키텍처와 통합하여 하나의 소프트웨어 아키텍처를 재구성하게 된다.

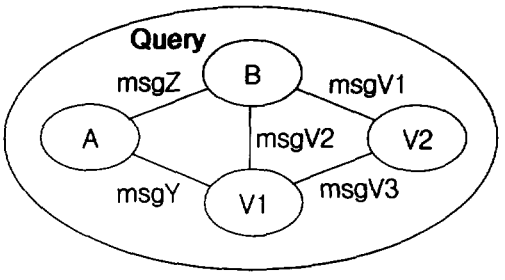
## 4. 실험 및 평가

이 실험에서는 제안한 통합된 구조의 아키텍처 재구성과 명세를 위한 모듈들의 기능성을 평가하기 위해 다양한 형태의 아키텍처를 정의하고, 정의한 아키텍처들과 유사한 형

태의 아키텍처를 정의하여 저장소에 저장하였다. 아키텍처 명세는 객체 지향 데이터베이스를 사용하여 저장하였다.

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.

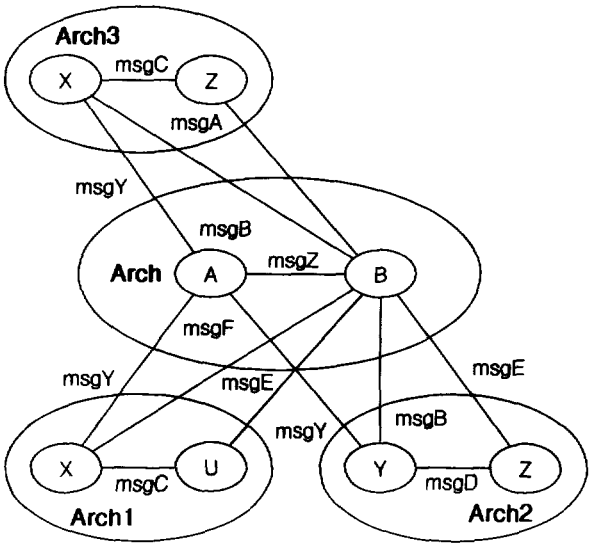
최종적으로 설계된 새로운 소프트웨어 아키텍처는 기존의 아키텍처와 통합된 구조로 재구성된다. (그림 4)는 검색을 위한 소프트웨어 아키텍처 명세 질의의 구조를 나타내고 있다.




(그림 4) 실험에 사용된 아키텍처 명세 질의

소프트웨어 아키텍처 검색에 사용한 질의는 단순 컴포넌트 2개와 복합 컴포넌트 2개, 그리고 메시지 커넥터 2개와 가변 커넥터 3개로 구성된 단순한 아키텍처 명세의 형태를 가지고 있다. 버전 검색기는 단순 컴포넌트와 메시지 커넥터가 동일하며, 가변 커넥터의 메시지 종류가 일치하는 소프트웨어 아키텍처 명세 버전을 저장소에서 검색해야 한다.

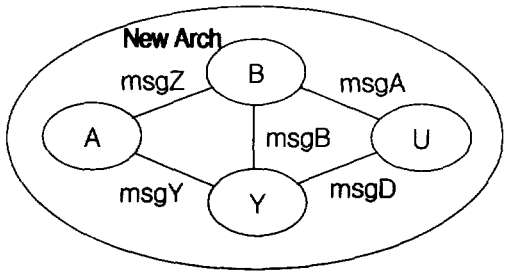
(그림 5)는 개발자가 선택한 저장소의 소프트웨어 아키텍처의 명세 구조를 보여주고 있다.




(그림 5) 검색된 기존 아키텍처 명세 구조

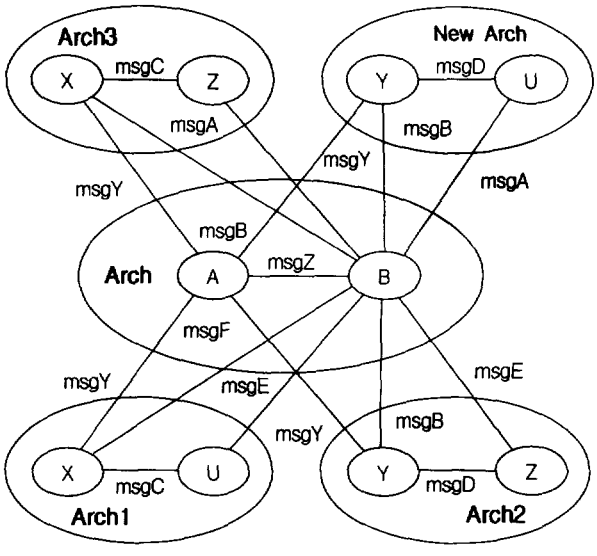
개발자는 기존의 재구성 소프트웨어 아키텍처 명세가 포함하고 있는 각각의 아키텍처를 참조하고, 이를 재사용하여 자신의 개발 환경에 적합한 새로운 소프트웨어 아키텍처를 설계하게 된다.

(그림 6)은 개발자가 기존 아키텍처를 재사용하여 새로 정의한 아키텍처 명세 구조를 보여주고 있다.




(그림 6) 개발자가 새로 정의한 아키텍처 명세 구조

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 7) 통합된 구조의 아키텍처 명세 구조

(그림 7)은 검색 모듈에 의해 검색된 기존의 아키텍처와 개발자의 새로운 아키텍처가 통합된 구조로 재구성된 아키텍처 구조를 나타내고 있고, <리스트 2>는 통합된 구조로 재구성된 소프트웨어 아키텍처의 연관 관계와 포함 관계 정보를 기반으로 XML 형태의 명세 구조로 표현된 아키텍처 명세의 일부를 나타내고 있다.

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

〈리스트 2〉 통합된 재구성 아키텍처 명세의 일부

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

## 5. 결 론

본 논문에서는 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 제공한다. 제안된 방식은 아키텍처 관리를 위한 기본적인 분류를 제공함으로써 재사용을 위한 검색의 효율을 높일 수 있으므로 재사용성 향상을 위한 기반을 제공할 수 있다.

제안된 명세 구조는 각 버전별로 독립된 저장 공간과 구조를 사용하는 기존 아키텍처 관리 방식들의 저장 공간 활용도와 검색에서의 비교 대상을 증가시킴으로써 시간 복잡성을 증가시키며 정확성을 감소시키는 문제를 해결하였다. XML기반의 아키텍처 명세 구조를 사용하여 통합된 구조의 아키텍처를 관리함으로써 설계 단계에서의 구조적인 재사용이 가능하다. 그리고, 재사용 과정에서 발생하는 아키텍처의 변경 내용을 명세하기 위해 보다 유연성 있고 효율적인 방법을 제공할 수 있다.

현재, 아키텍처를 보다 쉽게 직관적으로 표현할 수 있는 다이어그래밍 도구의 개발과 아키텍처 검색을 통한 재현율과 정확성 향상 평가에 관한 연구를 진행하고 있으며, 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.

## 참 고 문 헌


[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 Reusc, 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] G. Abowd, R. Allen and D. Garlan, "Using style to understand descriptions of software architecture," Proceedings of SIGSOFT '93 : Foundations of Software Engineering, Software Engineering notes, Vol.18, No.5, pp.9-20, Dec., 1993.

[6] Yhomas R. Dean and Jamcs R. Cordy, "A Syntactic Theory of Software Architecture," IEEE Transactions on Software Engineering, Vol.21, No.4, pp.302-313, Apr., 1995.

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

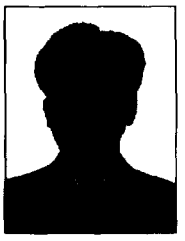
[8] W. Rossak, V. Kirova, L. Jololian, H. Lawson and T. Zemel, "A Generic Model for Software Architectures," IEEE Software, pp.84-92, Jul./Aug., 1997.

[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] 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.

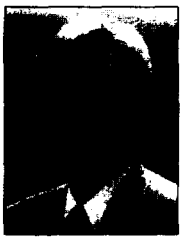
[11] N. Me128dvidovic, D. S. Rosenblum and R. N. Taylor, "A Language and Environment for Architecture-Based Soft-

ware Development and Evolution," Proceedings of the 21st International Conference on Software Engineering, May, 1999.
[12] P. Clements and P. Kogut, "The Software Architecture Renaissance," The Journal of Defense Software Engineering, Vol.7, 1994.
[13] 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.
[14] Shawn Butler, David Diskin, Norman Howes and Kathleen Jordan, "Architectural Design of a Common Operating Environment," IEEE Software, pp.57-65, Nov., 1996.
[15] 김상길, 이윤수, 윤경섭, 왕창종, "XML 기반 소프트웨어 컴포넌트 명세 및 검색", 한국정보처리학회, 추계학술발표논문집, 제6권 제2호, 1999년, 10월.
[16] 이윤수, 윤경섭, 왕창종, "재사용을 위한 XML 기반 소프트웨어 아키텍처 명세 언어", 한국정보처리학회, 정보처리논문지, 제7권 제3호, 2000년, 3월.
[17] R. Conradi and B. Westfechtel, "Version Models for Software Configuration Management," ACM Computing Surveys, Vol.30, No.2, pp.232-282, Jun., 1998.



**안 치 돈**

e-mail : joheunid@selab.inha.ac.kr
1995년 인하대학교 전자계산공학과(학사)
1997년 인하대학교 대학원 전자계산공학과 (공학석사)
1998년 현재 인하대학교 대학원 전자계산공학과(박사과정)
2000년 현재 ㈜인터정보 기술연구소 재직
관심분야 : XML, CBSE, 재사용, 멀티미디어 구조



**왕 창 종**

e-mail : cjwangse@inha.ac.kr
1964년 고려대학교 물리학과 학사
1975년 성균관대학교 경영학과 석사
1979년~현재 인하대학교 전자계산 공학과 교수
관심분야 : 소프트웨어공학, 분산객체컴퓨팅, 원격교육