재사용을 위한 XML 기반 소프트웨어 아키텍쳐 명세 언어

이 윤 수† · 윤 경 섭†† · 왕 창 종†††

요 약

재사용을 고려한 컴포넌트 명세 언어는 컴포넌트 분류, 검증 및 검색에 가장 기본적인 조건이다. 기존에 이미 많은 명세 언어가 사용되어 왔지만 이들의 대부분은 구현을 고려한 명세언어로써 명세의 복잡성과 불필요한 요소가 많았다.

본 논문에서는 이러한 문제점들을 해결하기 위해 XML기반의 컴포넌트 명세와 소프트웨어 아키텍처 명세 언어를 제안한다. 제안한 명세언어는 시그니쳐 명세, 기능 명세, 메시지 명세로 구성된 컴포넌트 명세와 XML 기반의 텍스트 표기법과 그래픽 표기법을 제공하는 소프트웨어 아키텍쳐 명세로 구성된다. 기능 명세를 통해 행위 일치 검색을 지원하고, XML을 통한 문서관리 및 검색의 효율성을 향상시킬 수 있으며, 컴포넌트의 블랙박스 재사용을 지원한다. 소프트웨어 아키텍쳐 명세는 메시지 기반의 아키텍쳐 명세를 통해 아키텍처의 구조적 재사용, 즉 화이트 박스 재사용을 지원한다.

XML based Software Architecture Specification Language for Reuse

Yoon-Soo Lee† · Kyung-Seob Yoon†† · Chang-Jong Wang†††

ABSTRACT

Component specification languages in consideration of reuse are essential factor in classification, verification and retrieval of components. A number of legacy specification languages have already been used, however, they are complex and include many unnecessary elements in the specification for implementation.

In this paper, we present XML-based component specification and software architecture specification language to solve these problems of legacy specification languages. The presented specification languages consist of component specification, which is composed of signature specification, interface specification and message specification, and software architecture specification providing graphical notations and textual notations. Component specification supports component retrieval with behavioral match and black-box reuse of component. In addition to this, it improves the efficiency of retrieval and document management with XML-based component specification. Software architecture specification supports the structural reuse of architecture, which is white-box reuse, through message-based architecture specification.

† 정 회 원 : 안산공과대학 전산정보과 교수
†† 정 회 원 : 인하공업전문대학 컴퓨터정보과 교수
††† 정 회 원 : 인하대학교 전자계산공학과 교수
논문접수 : 1999년 10월 27일, 심사완료 : 2000년 3월 6일

1. 서 론

세계적인 소프트웨어 시장에서 소프트웨어 개발에서의 주된 변화는 규모가 큰 소프트웨어 시스템을 개발, 분류, 갱신하는 방식으로 이루어지고 있다[1]. 소프트웨어 시스템을 하나의 컴포넌트 조합으로 설계하고 구현하는 것은 이제 더 이상 새로운 개념이 아니며 규모가 큰 소프트웨어 개발은 컴포넌트의 적절한 검색, 검증, 그리고 통합 과정을 통해 이룰 수 있게 되었다.

이러한 컴포넌트들의 재사용을 위해 우선 필요한 것은 컴포넌트의 기능을 명세하는 명세 언어이다 컴포넌트 명세에 있어 기존에 이미 많은 명세 언어가 사용되고 있으며, 이들은 나름대로의 장점을 가지고 있다. 그러나 이들은 대부분 구현을 위해 특정 언어에 의존성이 강하며, 표현 능력에 한계를 가지고 있다[2] 이러한 한계를 해결하기 위해 분재 요구사항에 대한 설명, 컴포넌트의 기능, 그리고 컴포넌트 구조에 대한 명확한 정의를 제공하는 정형화된 명세 언어(Formal Specification Language)에 대한 연구가 현재 이루어지고 있다[3].

그러나 기존의 명세 언어 대부분들은 구조적 프로그래밍 방식을 고려한 형태의 명세있기 때문에, 재사용을 고려한 컴포넌트 명세에 적용하기에는 부족하다 또한 이러한 명세는 설계단계에 초점을 두고 있기 때문에 소프트웨어 재사용을 위해서는 보완이 필요하다.

본 논문에서 제안한 명세 언어는 컴포넌트 기능 명세와 소프트웨어 아키텍처 명세 두 형태의 명세 언어를 제공한다 두 형태의 명세는 서로 개별적인 것이 아니라 컴포넌트 명세 언어로 기술된 컴포넌트를 소프트웨어 아키텍처 명세에서 사용할 수 있다. 그리고 제안된 명세 언어를 XML(eXtensible Markup Language)형태로 사상(mapping) 시킴으로써 컴포넌트 명세 언어에 대한 특정 지식 없이 원하는 컴포넌트에 대한 요구 사항 정보를 기술할 수 있도록 한다. 그리고 기술된 명세는 XML 문서로 저장되므로 효율적으로 관리할 수 있고, 웹 상에서 원하는 정보를 효율적으로 검색할 수 있다

2. 소프트웨어 아키텍처 및 기존의 명세언어

2.1 소프트웨어 아키텍처

최근 소프트웨어 공학에서의 연구 경향은 모듈 인터페이스 언어, 특정 영역 아키텍처(specific domain architecture), 구조적 기술 언어(structured description language), 설계 패턴(design pattern), 구조적 설계를 위한 기초, 그리고 구조적 설계 환경 영역에 대해 활발한 연구가 이루어지고 있다[4]. 소프트웨어 아키텍처는 프로그램과 시스템 컴포넌트, 그들의 상호관계에 대한 구조, 그리고 컴포넌트를 설계하고 개발하기 위한 규칙 및 지침서라고 정의한다[5, 6].

소프트웨어 아키텍처에 관련된 최근 두 가지 경향은 복잡한 소프트웨어 시스템을 구조화하기 위해 메소드(method), 기술(description), 패턴(pattern), 그리고 특징(idioms)을 공유하는 것과 사용 가능한 프레임워크(framework)를 제공하기 위한 특정 영역의 이용 문제에 관한 것이다[6]

소프트웨어 개발에서 정형화된 소프트웨어 구조를 사용하게 되면 시스템 설계에서 고수준의 추상화가 가능하여 복잡한 시스템에 대한 이해를 돕고, 컴포넌트 라이브러리를 통한 다중 레벨 재사용이 가능하다 그리고, 시스템 구현 시 소프트웨어 시스템의 기본적인 요구사항과 기대되는 발전 방향에 대한 시뮬레이션을 통한 요구사항과 발전 범위의 축적을 통해 소프트웨어 개발에 있어서 위험성을 감소시킨다

2.2 기존의 소프트웨어 아키텍처 명세 언어

메시지 기반의 소프트웨어 아키텍처 명세 언어로는 Rapide와 C2 등이 있고, 두 계층 형태를 통해 언어에 종속적인 형태와 언어에 독립적인 형태를 제공하는 Larch 언어 등이 있다. Rapide는 분산 시스템 구조의 프로토타이핑(prototyping)을 위해 설계된 이벤트 기반의 동시 객체 지향 언어(Event-based Concurrent Object-Oriented Language)이다[7] Rapide에서 아키텍처는 모듈에 대한 명세 집합, 인터페이스간 직접적인 통신을 정의하는 연결 규칙의 집합, 그리고 통신 패턴을 정의하는 정형화된 제약조건으로 구성된다[8].

C2는 메시지 기반 스타일을 사용하여 사용자 인터페이스를 기술하는 언어이다[9] C2의 구조적 스타일에서 주된 두 가지 요소는 컴포넌트(components)와 커넥터(connectors)인데, 이는 아키텍처를 구성하는 요소이기도 하다. 컴포넌트는 커넥터를 통해서만 통신할 수 있으며, 하나의 커넥터에 연결될 수 있는 컴포넌트, 또는 커넥터의 수에는 제한이 없다. C2에서 컴포넌트간 통신은 메시지 전달을 통해 이루어지며, 커넥터는 컴포넌트에서 발생한 이벤트를 다른 컴포넌트에 전달한다.

Larch 언어는 양질의 소프트웨어 유지보수 및 개발

을 돕기 위한 명세 언어의 일종으로 두 계층 형태를 통해 언어에 종속적인 형태와 언어에 독립적인 형태를 제공한다[2]. Larch는 컴포넌트와 각 프로그래밍 언어 사이의 인터페이스 기술을 위한 인터페이스 언어(Larch Interface Language)와 언어 독립적인 공유 언어(Larch Shared Language)의 2계층(two-tired hierarchy)의 구조를 가진다.

그러나 Larch와 같은 두 계층의 형태는 언어에 독립적이기 때문에 컴포넌트의 특징을 표현하는데 불필요한 요소를 많이 가진다. 기존의 명세 언어는 많은 경우 정형 명세에 기반하기 때문에 정형 기법에 관한 지식이 부족한 개발자의 경우 명세에 어려움이 있다. 그리고 대부분의 명세 언어들은 기능(functionality)에 관한 명세로 비기능의 명세 지원이 미흡하고, 특정 언어에 종속적이며, 명세되는 시스템의 관점이 제한적이다.

이러한 명세 언어를 컴포넌트 명세에 사용하기 위해서는 다음과 같은 문제점들을 보완해야 한다. 첫째, 기존의 명세 언어는 구조적인 언어로 작성된 함수단위의 기술을 제공하기 때문에 컴포넌트가 외부와 연결을 제공하는 인터페이스에 의해 기능이 정의되는 특징을 표현하기에 부족하다. 둘째, 기존의 명세 언어는 재사용을 고려한 목적으로 설계되지 않았기 때문에 특정 지식이 없는 개발자가 원하는 컴포넌트를 검색하기에는 많은 어려움이 있다

컴포넌트 기반 소프트웨어공학의 가장 큰 목적은 기존 객체 지향 기술이 해결하지 못한 재사용성에 있다. 재사용성을 높이기 위해서는 명세 언어가 재사용을 기반을 두고 정의되어야 한다

따라서, 본 논문에서는 명세에 대한 특성 지식이 없는 개발자들도 그래픽 표기법을 통해 쉽게 명세할 수 있고, 컴포넌트의 재사용성을 향상시키기 위한 XML에 기반한 새로운 컴포넌트 명세 언어를 제안하였다. 또한 컴포넌트의 특징을 표현하기 위해 언어 종속적인 형태를 제거하고 언어 독립적인 형태만을 제공할 수 있게 하였다.

2.3 XML

XML은 주로 WWW(World Wide Web) 상에서 쉽고 일관된 방법으로 데이터를 형식화하며, 전송을 위한 구조적 데이터 형식을 제공하는 메타-마크업(meta-markup) 언어이다[10]. 특히 XML은 웹 상에서의 사용에 목적을 두고 설계된 SGML(Standard Generalized Markup Language)의 부분집합 형태로 제공된다. 이는 여러 플랫폼 상에서 내용을 좀더 정확하게 선언하고 의미있는 검색 결과를 얻을 수 있도록 하며, 새로운 웹 기반 데이터를 보여주고 처리하는 응용으로 동작 가능하다

XML 문서는 DTD(Document Type Definition)와 결합되어 있고, DTD의 규칙에 따르면 대부분의 문서 표현이 가능하다. DTD는 XML 문서에서 사용할 태그를 정의하고, 이들이 어떤 순서로 동작하는지를 정의할 수 있다[11]

XML에서 사용자는 태그(tag)들을 제한없이 정의할 수 있고, 정의한 태그들에 구조적 데이터를 표시하는 프레임워크를 제공한다 구조적 데이터는 사용자, 또는 응용에 독립적이고 단순화된 형태로 정의한다. 그리고 XML은 구조적 데이터와 사용자 인터페이스를 분리한다[11]. XML에서는 브라우지에서 데이터를 나타내기 위해 XSL(eXtensible Style Language)과 CSS(Cascading Style Sheets)와 같은 스타일시트를 사용한다.

본 논문에서는 XML을 소프트웨어 아키텍처 명세에 사용하여 개발자로 하여금 웹 상에서 원하는 정보를 쉽게 검색할 수 있도록 아키텍처 명세를 XML 형태로 사상시켜 저장, 관리를 용이하게 하였다.

3. 메시지 이론

아키텍처 명세는 구조적인 재사용을 목적으로 하고 있으므로 구조적인 동등에 관한 연구가 요구된다. 이 장에서는 본 논문에서 커넥터로 사용되는 메시지에 관한 인산 이론과 메시지 일치에 관한 정의 등을 제안한다.

정의 1. 완전 동등(Perfect Equivalence)

컴포넌트들이 다른 컴포넌트에게 발생시키는 커넥터인 메시지 m_i와 m_j가 다음 조건을 만족할 때 이를 완전 동등이라 한다

$$(i)\ Name(m_i) = Name(m_j)$$
$$(ii)\ Count(m_i.parameter) = Count(m_j.parameter)$$
$$(iii)\ Type(m_i.parameter_k) = Type(m_j.parameter_k)$$
$$(iv)\ Act(m_i) = Act(m_j)$$

완전동등은 $m_i =_p m_j$로 표기한다. 단, $1 \le i, j, k \le n$이며, Count는 파라미터의 개수를 세는 함수, Type은 파라미터의 자료형을 확인하는 함수, Act는 메시지에

대한 동작을 의미한다 정의 1의 (i)은 메시지들의 함수명이 같은 경우이고, (ii)는 메시지들의 파라미터 개수가 같은 경우이다. (iii)은 메시지들의 파라미터 자료형이 같은 경우이며, (iv)는 메시지들의 행위 결과가 같은 경우이다.

정리 1. 동등의 특성

$(i)(\forall message\ m)m == m$
$(ii)(\forall message\ m_i, m_j)m_i == m_j \rightarrow m_j == m_i$
$(iii)(\forall message\ m_i, m_j, m_k)(m_i == m_j \wedge m_j == m_k) \rightarrow m_i == m_k$

메시지의 동등은 정리 1의 (i)과 같이 '=='로 표기하며, 두 메시지가 동일한 메시지임을 나타낸다. 정리 1의 (ii)는 메시지 동등의 대칭성을 표현하고 있다. 그리고 (iii)은 메시지 m_i와 m_j가 동등하고 메시지 m_j와 m_k가 동등하면, 메시지 m_i와 m_k는 동등하다는 이행성(transitivity)의 특징을 표현한다.

정의 2. 강한 동등(Strong Equivalence)

메시지 m_i와 m_j가 다음 조건을 만족할 때 이를 강한 동등이라 한다.

$(i)Count(m_i.parameter) == Count(m_j.parameter)$
$(ii)Type(m_i.parameter_k) == Type(m_j.parameter_k)$
$(iii)Act(m_i) == Act(m_j)$

강한 동등은 $m_i ==_{st} m_j$로 표기하며, 기능적인 완전 동등을 의미한다.

정의 3. 시그니쳐 동등(Signature Equivalence)

메시지 m_i와 m_j가 다음 조건을 만족할 때 이를 시그니쳐 동등이라 한다

$(i)Count(m_i.parameter) == Count(m_j.parameter)$
$(ii)Type(m_i.parameter_k) == Type(m_j.parameter_k)$

시그니쳐 동등은 $m_i ==_{sg} m_j$로 표기한다.

정의 4. 구조 동등(structure equivalence)

메시지 m_i와 m_j가 다음 조건을 만족할 때 이를 구조 동등이라고 한다.

$(i)Count(m_i.parameter) == Count(m_j.parameter)$
$(ii)reoder(Type(m_i.parameter_i)) == reorder(Type(m_j.parameter_k))$

구조 동등은 $m_i ==_{struct} m_j$로 표기한다 정의 4의 (i)은 메시지 m_i와 m_j의 파라미터 개수가 같은 경우이며, (ii)는 메시지 m_i와 m_j의 파라미터 자료형을 재배열할 경우 같게 되는 경우이다.

정의 5. 시간 동등

메시지 m_i와 m_j가 다음 조건을 만족할 때 이를 시간 동등이라고 한다.

$$m_i.starttime = m_j.starttime$$

시간 등등은 $m_i ==_t m_j$로 표기하며, 정의 5는 메시지 m_i와 m_j에서 보내는 메시지 전송 시간이 같은 경우이다

정리 2 시간 순서

$(i)(\forall\ message\ m)m \leq_t m$
$(ii)(\forall\ message\ m_i, m_j)(m_i \leq_t m_j \wedge m_j \leq_t m_i) \rightarrow m_i ==_t m_j$
$(iii)(\forall\ message\ m_i, m_j, m_k)(m_i \leq_t m_j \wedge m_j \leq_t m_k) \rightarrow m_i \leq_t m_k$

정리 2는 메시지의 시간순서에 관한 정리이다. 정리 2의 (i)은 메시지 시간순서의 표현이고, (ii)는 메시지 시간동등의 특성을 표현하고 있다. 그리고 (iii)은 메시지 시간 순서의 이행성을 표현하고 있다

4. XML 기반 명세 언어

명세 언어는 아키텍처 명세와 컴포넌트 명세 두 부분으로 구성된다. 제안하는 컴포넌트 명세 언어의 특징은 인터페이스에 기반한 시그니쳐 명세와 기능 명세를 모두 지원한다 이는 컴포넌트의 재사용을 위한 시그니쳐 일치 검색, 행위 일치 검색 방법을 모두 지원할 수 있게 한다.

4.1 XML 기반 컴포넌트 명세

본 논문에서 제안하는 명세 언어는 XML을 기반으로 하고 있다 이를 통해 명세 자체의 검색에서 기존의 명세 방법보다 검색의 효율 및 수준을 높일 수 있으며, 문서 구조가 논리적으로 명확하여 명세 파악의 신뢰도와 속도 향상을 가져올 수 있었다. 또한 구조화된 명세는 새로운 정보의 추가 및 기존정보의 수정도 용이한 장점을 가지게 된다.

(1) 시그니쳐 명세

본 논문에서 제안한 명세 언어에서 시그니쳐는 컴포넌트의 IDL 명세 형태를 따른다. 시그니쳐 명세는 인터페이스의 입력 타입과 출력 타입을 기술함으로써 특정 타입을 검색하는 시그니쳐 검색 방법을 제공한다.

시그니쳐 명세는 <sig_list> 태그에 포함된 <require> 태그 부분에 기술한다. 전체 인터페이스의 개수를 기술하기 위한 <num_of_sig> 태그가 있으며, 인터페이스 이름을 기술하는 <name> 태그가 <signature> 태그에 포함되어 있다 인터페이스의 이름에는 특별한 규칙이 없으며, 다만 이름 지정 시 가능성을 고려하면 핵심어 검색방법에서 유용하게 사용될 수 있다 <parameter> 태그는 <type>, <name>, 그리고 <direct> 태그를 포함하고 있는데, 파라미터 데이터 타입을 정의하는 <type> 태그는 기본적으로 IDL데이터 타입에 기반을 둔다. 타입은 크게 기본 타입과 생성 타입으로 구분한다 <direct> 태그는 파라미터가 입력(in)인지 출력(out)인지 입출력(inout)인지를 지정하는 태그를 나타내고, <name> 태그는 파라미터의 이름을 기술하는 부분이다.

(2) 기능 명세

기능 명세는 인터페이스에 의해 제공되고 접근가능한 컴포넌트 기능의 관점에서 명세한다. 따라서 기능 명세는 인터페이스의 기능 명세 형태를 따른다 기능 명세 부분은 <signature> 태그에 포함되어 있는데, <modifies> 태그와 <ensure> 태그에 구체적인 인터페이스의 기능을 기술한다. <modifies> 태그에는 인터페이스 기능 수행 중 변화가 일어나는 변수를 기술하고, <ensure> 태그에는 해당 인터페이스의 실제 동작에 관해 명세한다. <ensure> 태그 안의 명세는 자바 언어의 제어 구분과 자료형에 따라 기술한다

(3) 메시지 명세

각 컴포넌트는 요청하고 처리할 수 있는 메시지 목록 명세를 포함한다. <msg_list> 태그를 시작으로 컴포넌트가 가지고 있는 메시지의 개수를 나타내는 <num_of_msg> 태그가 있으며, <message> 태그에 각각의 메시지에 대한 구체적인 기술을 한다. <name> 태그에 메시지의 이름을 기술하며, <param_list> 태그에 메시지의 파라미터에 대한 기술을 한다. <num_of_param> 태그에는 파라미터의 개수를 기술하고, <parameter> 태그에 파라미터의 타입과 이름, 그리고 파라미터의 방향을 기술한다

4.2 XML 기반 컴포넌트 명세의 구조

제안한 컴포넌트 명세의 전체 구조는 <리스트 1>과 같다.

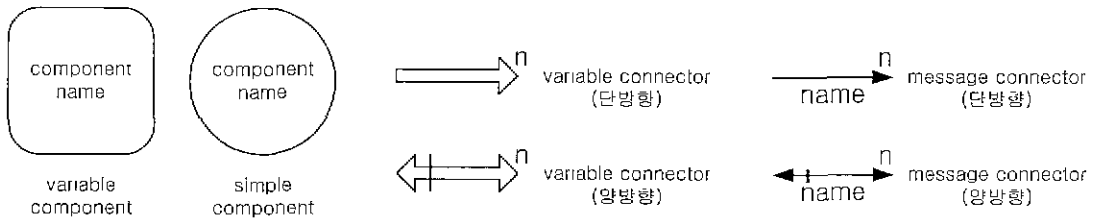
<리스트 1> 컴포넌트 명세의 전체 구조

```
<component>
 <type></type> <name></name>
 <sig_list> <num_of_sig></num_of_sig>
<signature> <return_type></return_type>
<name></name>
 <require>
 <param_list> <num_of_param></num_of_param>
 <parameter> <type></type> <name></name>
 <direct></direct>
  </parameter>
 </param_list>
 </require>
 <modifies></modifies>
 <ensure></ensure>
 </signature>
 </sig_list>
 <msg_list> <num_of_msg></num_of_msg>
 <message> <name></name>
 <param_list> <num_of_param></num_of_param>
<parameter> <type></type> <name></name>
<direct></direct></parameter>
 </param_list>
 </message>
 </msg_list>
</component>
```

<component> 태그를 시작으로 각각의 하위 태그들을 계층적으로 표현함으로써 개발자가 그 의미를 파악하고 데이터를 입력하는데 어려움이 없도록 표현하였다. 기존 명세 언어의 복잡성을 피하고 XML의 특징인 구조적인 데이터 표현을 명세 언어 기술에 활용함으로 개발자가 입력기 인터페이스에 기술하고자 하는 데이터만을 입력하면 XML 문서는 기본 구조에 따라 자동으로 생성된다. 따라서 XML 문서를 사용하여 데이터를 표현할 때 얻게 되는 데이터 관리의 일관성을 그대로 유지할 수 있고, 기본적으로 제공되는 뷰와 DTD를 통해 오류 발생률을 감소시킬 수 있다.

4.3 소프트웨어 아키텍쳐 명세

소프트웨어 아키텍쳐는 전체 시스템의 컴포넌트와 컴포넌트 사이의 상호작용과 연결을 조절하는 커넥터



(그림 1) 아키텍쳐 명세의 그래픽 표기

를 중심으로 분석하고 기술함으로써 시스템의 구조를 명세한다. 본 논문에서는 소프트웨어 아키텍처도 하나의 컴포넌트라는 관점으로 명세에 접근하고 메시지 기반의 아키텍쳐 명세를 제시한다 소프트웨어 아키텍쳐 명세 언어는 구조적인 재사용, 즉 화이트박스 재사용을 지원하고, 아키텍쳐 명세를 XML 형태로 사상시킴으로 저장 관리를 용이하게 한다

소프트웨어 아키텍처 명세 언어의 그래픽 표기에 사용되는 요소들은 (그림 1)에 나타내었다. (그림 1)의 양방향 커넥터를 나타내는 표기법에서 직선을 포함하는 쪽의 화살표는 메시지를 발생시키는 컴포넌트로 부터의 방향성을 나타낸다.

(1) 가변 컴포넌트(variable component)

가변 컴포넌트는 일반적인 변수 역할을 한다. 가변 컴포넌트는 아키텍처 기술을 위해 사용될 수 있는데, 기술한 아키텍처에서 가변 컴포넌트는 임의의 컴포넌트로 대치되거나 복합 컴포넌트(composite component)로 대치될 수 있다. 가변 컴포넌트의 텍스트 표기는 <리스트 1>의 <component> 태그에 포함되어 있는 <type> 태그에 Variable이라 기술함으로써 정의할 수 있고, <name> 태그에는 컴포넌트 이름을 정의한다

(2) 단순 컴포넌트(simple component)

소프트웨어 아키텍처 명세에서 특정 아키텍처나 컴포넌트에 의해 기능을 가지는 경우 필요한 기능을 지니는 아키텍처나 컴포넌트 명세를 위해 사용한다. 이 명세의 텍스트 표기는 가변 컴포넌트에서의 설명과 유사한 방법인 <component> 태그의 <type> 태그에 Simple이라고 기술함으로써 정의할 수 있다. 가변 컴포넌트와는 달리 기능 명세가 요구되므로, 기능 명세를 위해 컴포넌트 명세의 내부에 직접 기능명세를 하는 경우에는 4.1절의 컴포넌트 기능 명세를 사용한다

커넥터는 메시지를 의미하며, <connector> 태그를 사용한다. 커넥터의 종류에는 메시지 커넥터와 가변 커넥터가 있다 각 커넥터는 방향성을 지니는데 커넥터의 방향성에 따라 분류하면 단방향 커넥터, 양방향 커넥터로 구분할 수 있다.

(3) 가변 커넥터(variable connector)

가변 커넥터는 일반적으로 가변 컴포넌트로 부터 발생되는 메시지를 의미한다. 가변 커넥터는 구조적 재사용을 위한 요소이다. 가변 커넥터는 사용자가 컴포넌트에서 발생되는 메시지에 대한 정확한 지식이 없을 때 메시지의 방향과 타입 정도를 지정할 수 있도록 정의된다 저장소에 저장될 때에는 특정한 메시지 커넥터로 변환되어 저장된다. 이러한 가변 커넥터는 사용자의 질의를 표현할 때 유용하게 사용될 수 있다.

가변 커넥터의 텍스트 표기는 <connector> 태그가 포함하고 있는 <type> 태그에 Variable이라고 기술함으로써 정의하고, 커넥터의 방향성을 기술하기 위한 <direct> 태그, 메시지의 출력 타입을 나타내기 위한 <return_type> 태그, 메시지 이름을 기술하는 <name> 태그, 메시지 파라미터에 대한 구체적인 기술을 위해 <param_list> 태그를 기본값으로 가진다 그리고 커넥터는 메시지를 발생시키는 컴포넌트나 메시지를 처리할 컴포넌트를 명세하는데, 발생시키는 메시지는 <src> 태그를 사용하고 메시지를 받아 처리하는 컴포넌트는 <dest> 태그를 사용하여 명세한다

(4) 메시지 커넥터(message connector)

메시지 커넥터는 단순 컴포넌트에서 발생하는 특정한 메시지를 의미한다. 메시지 커넥터를 이용하여 명세한 소프트웨어 아키텍처는 특정 기능을 수행하는 복합 컴포넌트를 표현한다 메시지 커넥터의 텍스트 표기는 <type> 태그에 Simple이라고 기술함으로써 정의

할 수 있으며, 기본적으로 가변 커넥터와 같은 구조를 지닌다. 기능적인 차이점을 제외하면 같은 방식으로 표기한다.

4.4 소프트웨어 아키텍처 명세의 구조

제안한 소프트웨어 아키텍처 명세의 전체 구조는 <리스트 2>와 같다. <component> 태그 부분은 <리스트 1>에서 정의한 컴포넌트 명세의 전체 구조를 그대로 사용한다

<리스트 2> 아키텍처 명세의 전체 구조

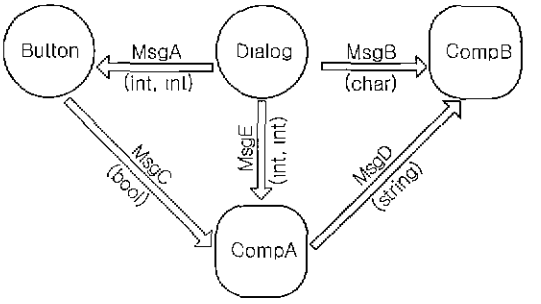
```
<component_architecture>
<component></component>
<connector>
 <type></type> <direct></direct> <return_type>
 </return_type> <name></name>
 <param_list> <num_of_param></num_of_param>
<parameter> <type></type> <name></name>
<direct></direct> </parameter> </param_list>
 <src></src>
 <dest></dest>
</connector>
</component_architecture>
```

5. 실험 및 평가

이 장에서는 소프트웨어 아키텍처 명세를 위한 도구와 컴포넌트 검색 시스템 프로토타입을 구현하여 4장에서 제안한 명세 언어의 타당성과 검색에 적합함을 보이기 위해 질의가 XML로 표현됨을 보이고, 정확성과 재현율을 측정하였다.

우선 개발자는 개발에 필요한 컴포넌트나 소프트웨어 아키텍처를 검색하기 위해 질의를 작성한다. 구조와 컴포넌트들이 저장소에 저장되어 있을 때, 제안한 검색 구조에서는 우선 개발자의 질의와 일치하는 구조를 검색하게 되고, 소프트웨어 재구성을 위해 시그니처 일치 방법을 사용하여 컴포넌트를 검색한 후, 검색된 컴포넌트를 대상으로 행위 일치에 의한 방법을 적용하여 적합한 소프트웨어 아키텍처와 컴포넌트를 검색한다. (그림 2)는 실험에서 사용될 개발자의 질의 명세를 그래픽 표기법으로 나타낸 것이다

(그림 2)에서 명세는 4개의 컴포넌트와 5개의 커넥터로 구성되어 있다. Dialog 컴포넌트와 Button 컴포넌트에 대해서는 아키텍처 재구성에 필요한 컴포넌트이며, CompA와 CompB는 아키텍처에 포함될 컴포넌트

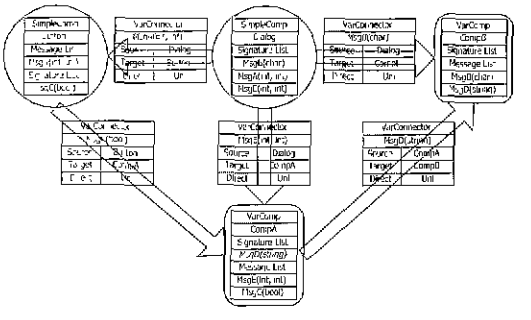


(그림 2) 아키텍처 검색을 위한 질의 명세

들이다. Dialog 컴포넌트는 Button 컴포넌트와 CompA 컴포넌트에 메시지 커넥터로써 (int, int)형의 메시지를 전달하고, CompB 컴포넌트에 char형의 메시지를 전달한다 Button 컴포넌트는 CompA 컴포넌트에 메시지 커넥터로써 bool 형의 메시지를 전달하고, CompA 컴포넌트는 CompB 컴포넌트에 가변 커넥터로써 string 형의 메시지를 전달하고, CompB 컴포넌트로 부터 이에 대한 응답을 받는 질의이다

(그림 3)은 (그림 2)의 아키텍처 내부 자료 구조를 표현한 것이다 단순 컴포넌트인 Dialog에서는 3가지 타입의 메시지를 발생시키며, 임시로 정해진 MsgA는 Button을 타겟으로 갖는 가변 커넥터로 정의된다.

각각의 컴포넌트들은 커넥터로 연결되어 있고, 그래프 형태의 자료 구조이지만 각 커넥터에는 소스와 타겟을 명시함으로써 선형적인 형태로 관리가 가능하다. 각 요소들은 사용자가 질의에서 입력한 순서대로 리스트에 저장되며, 저장된 요소들은 저장된 형태 그대로 XML로 변환될 수 있다.



(그림 3) 아키텍처 리스트 형태의 내부 자료 구조

이 논문에서 제안한 소프트웨어 아키텍처 명세 언어

를 사용하여 실험에 사용된 아키텍처를 명세하였을 때 생성되는 XML 기반의 텍스트 표기는 <리스트 3>과 같다. <리스트 3>에서는 생성된 아키텍처 명세 중 단순 컴포넌트인 Button 컴포넌트와 Dialog 컴포넌트에 대한 명세와 (int, int) 형의 커넥터인 MsgA에 의한 두 컴포넌트간 메시지 전달 부분만을 나타내었으며, 구조적인 형태로 표현될 수 있는 컴포넌트와 1대 1로 사상됨을 알 수 있다.

<리스트 3> 아키텍처 질의의 텍스트 표기

```
<component_architecture> <name>Query</name>
 <component>
 <type>Simple</type><name>Button</name>
 <sig_list><num_of_sig>1</num_of_sig>
 ...
 </sig_list>
 <msg_list><num_of_msg>1</num_of_msg>
 <message><name>MsgA</name>
 <param_list><num_of_param>2</num_of_param>
  <parameter><type>int</type><name/><direct/>
  </parameter>
  <parameter><type>int</type><name /><direct />
  </parameter></param_list></message></msg_list>
 </component>
 <component>
 <type>Simple</type><name>Dialog</name>
 <sig_list><num_of_sig>3</num_of_sig>
 <signature><return_type />
 <name>MsgA</name>
 <require><param_list>
  <num_of_param>2</num_of_param>
  <parameter><type>int</type><name/><direct/>
  </parameter>
  <parameter><type>int</type><name/><direct/>
  </parameter></param_list></require></signature>
 ...
 </sig_list>
 <msg_list />
 </component>
 <connector>
 <type>Variable</type><direct>Uni</direct>
 <return_type /><name>MsgA</name>
 <param_list><num_of_param>2</num_of_param>
 <parameter><type>int</type><name /><direct />
 </parameter>
 <parameter><type>int</type><name /><direct />
 </parameter></param_list>
 <src>Dialog</src><dest>Button</dest>
 </connector>
 ...
</component_architecture>
```

작성된 질의는 아키텍처 저장소에 저장된 아키텍처를 검색한다. 저장소에 저장된 아키텍처가 설계한 컴포넌트 검색 시스템에 전달되면, 저장되어 있는 아키텍처와 질의의 컴포넌트 수, 그리고 커넥터의 수와 방향, 그리고 파라미터의 형식을 이용하여 비교하여 후보 아키텍처를 선정한다. 후보 아키텍처들 중에는 개발의 요구 사항과 일치하는 컴포넌트를 포함하는 아키텍처들이 검색될 수 있지만, 만약 일치하지 않으면 요구 사항과 일치하는 컴포넌트로 아키텍처를 재구성하기 위해 컴포넌트 검색 방법을 이용하여 컴포넌트들을 검색하여야 한다.

질의에서 CompA와 CompB 컴포넌트는 사용자 인터페이스들이 저장된 컴포넌트 저장소로부터 컴포넌트를 검색하여 후보 아키텍처의 컴포넌트들과 대체한다. 대체한 아키텍처가 정확하게 개발자가 요구하는 아키텍처일 경우에 소프트웨어 아키텍처 재구성에 사용한다.

컴포넌트의 검색은 기존의 컴포넌트 검색 방식인 시그니처 일치 검색 방식을 사용하였다. 컴포넌트 검색의 경우, 제안한 메시지 이론에 의해 시그니처의 방향성과 파라미터의 형식, 파라미터의 개수, 파라미터의 재구성을 통해 질의가 일치하는가를 검색하였다. 정확성과 재현율의 측정은 (그림 2)의 질의에 대해 아키텍처 검색을 수행한 후, 아키텍처에 필요한 컴포넌트를 검색한 결과와 컴포넌트 검색만을 수행한 결과를 비교하여 측정한다.

〈표 1〉 실험 결과

	Component Retrieval Only			with Architecture Retrieval		
	recall	prec	comp	recall	prec	comp
A	0.75	0.5	107	0.75	0.75	21
B	0.67	0.36	107	0.67	0.8	16
C	0.67	0.5	107	0.75	0.69	31
D	0.75	0.43	107	1	0.8	11
E	0.78	0.17	107	0.78	0.58	27
F	0.71	0.56	107	0.71	0.71	19
G	0.62	0.76	107	0.67	0.93	32

- recall = 검색된 컴포넌트 중 질의를 만족하는 컴포넌트 수 / 질의를 만족하는 컴포넌트 수
- prec(precision) = 검색된 컴포넌트 중 질의를 만족하는 컴포넌트 수 / 검색된 컴포넌트 수
- comp(comparison) : 일치 수행 횟수(아키텍처 검색 병행 시 아키텍처 일치 수행 회수 45 추가)

실험에 사용한 컴포넌트 명세는 MFC(Microsoft Foundation Class) 라이브러리의 사용자 인터페이스

컨트롤과 ActiveX 컨트롤을 XML 기반 명세 언어로 정의하여 사용하였다. 사용자 인터페이스 컴포넌트의 개수는 총 107개이며, 아키텍처 검색을 위해 일반적으로 사용되는 사용자 인터페이스 아키텍처 45개를 추가로 정의하여 아키텍처 저장소에 저장하였다. <표 1>은 실험 수행 결과를 컴포넌트 검색만 수행한 결과와 아키텍처 검색을 병행한 경우로 구분하여 정리한 것이다. 단, 컴포넌트의 검색을 위해서는 시그니쳐 일치 검색만을 수행하였다.

실험 결과, 아키텍처 검색과 컴포넌트 검색을 병행하여 검색할 컴포넌트의 개수를 줄임으로써 컴포넌트 검색만을 수행한 결과에 비해 정확성과 재현율을 높일 수 있었다.

6. 결 론

본 논문에서는 메시지 기반의 소프트웨어 아키텍처와 컴포넌트를 명세하기 위한 XML기반의 명세 언어를 제안하였다. 구현을 고려한 정형화된 명세 언어의 복잡성을 줄이고, 명세를 이용한 행위 일치 검색의 지원을 가능하게 하였다.

컴포넌트 명세를 XML로 정의함으로써 컴포넌트 명세 요소들은 계층적인 구조를 가지며, 개발자는 뷰를 통해 이 구조를 볼 수 있다. 또한 XML 문서의 유효성은 DTD에 사용된 태그들과 데이터 타입을 정의함으로써 유지된다. 이러한 XML의 장점을 통해 명세 문서 관리의 효율을 향상시켰고, 동시에 블랙박스 재사용을 지원하게 되었다. 메시지 커넥터 기반의 소프트웨어 아키텍처 명세는 XML 기반의 텍스트 표기법을 제공함으로써 명세의 편이를 도왔으며, 이러한 표기법을 제공함으로써 컴포넌트의 블랙박스 재사용뿐만 아니라 아키텍처의 구조적 재사용을 제공하였다. 또한 아키텍처 검색을 이용하여 아키텍처를 검색한 후, 아키텍처 재사용을 위해 컴포넌트를 검색하도록 함으로써 컴포넌트 검색의 범위를 제한하여 컴포넌트 검색의 정확성과 재현율을 향상시키고, 일치 수행을 위한 비교 횟수를 줄임으로써 검색 수행 시간이 감소되었다.

참 고 문 헌

[1] Alan W. Brown, "Foundations for Component-Based Software Engineering," http://computer.org/cspress/CATALOG/BP07718/preface.htm.

[2] J.V. Guttag, and J.J. Horning, "A Tutorial on Larch and LCL, a Larch/C Interface Language," Proceeding of VDM91:Formal Software Development Methods, S. Prehn and W.J. Toetenel (eds.), Delft, Oct. 1991.

[3] J.J. Jeng and B.H.C. Cheng, "A Formal Approach to Reusing More General Components," Proceedings of IEEE 9th Knowledge-Based Software Engineering Conference, Monterey, California, pp.90-97, Sep. 1994.

[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] 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.

[6] 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.

[7] D. Bryan, "Rapide-0.2 language and tool-set overview," Technical Note CSL-TN-92-387, Computer System Laboratory, Stanford University, Feb. 1992.

[8] Rapide Design, "Team, The Rapide-1 Architectures Reference Manual," Program Analysis and Verification Group, Computer System Laboratory, Stanford University, Oct. 1994.

[9] J.E. Robbins, E.J. Whitehead Jr., N. Medvidovic, and R.N. Taylor, "A Software Architecture Design Environment for Chiron-2 Style Architectures," Arcadia Technical Report UCI-95-01, University of California, Irvine, Janc, 1995.

[10] R. Khare, and A. Rifkin, "X Marks the Spot," 1997, http://www.cs.caltech.edu/~adam/papers/xml/x-marks-the-spot.html.

[11] W3C Recommendation 10-February-1998, "Extensible Markup Language (XML) 1.0," 1998, http://www.w3.org/TR/1998/REC-xml-19980210

이 윤 수

e-mail : yslee@ansantc.ac.kr

1988년 인하대학교 전자계산학과 졸업(학사)

1990년 인하대학교 대학원 전자계산학과(이학석사)

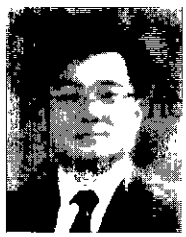
1997년 인하대학교 대학원 전자계산공학과(박사과정수료)

1995년~1996년 인하대학교 전자계산공학과 전임대우강사

1997년~현재 안산공과대학 전자계산소장

1996년~현재 안산공과대학 전산정보과 조교수

관심분야 : 컴포넌트 기반 소프트웨어 공학, 분산객체 컴퓨팅, 원격교육



윤 경 섭

e-maul : ksyoon@true.inhatc.ac.kr

1982년 인하대학교 전자계산학과 졸업(학사)

1984년 인하대학교 대학원 전자계산학과(이학석사)

1995년 인하대학교 대학원 전자계산학과(이학박사)

1995년~1999년 인하공업전문대학 전자계산소장

1987년~현재 인하공업전문대학 컴퓨터정보과 교수

관심분야 : 소프트웨어공학, 지능형 시스템, 분산시스템, 원격교육

왕 창 종

e-mail : cjwangse@inha.ac.kr

1964년 고려대학교 물리학과 졸업(학사)

1975년 성균관대학교(경영학석사)

1981년~1990년 인하대학교 전자계산소장

1992년~1993년 한국정보과학회 부회장, 전산교육연구회 위원장

1979년~현재 인하대학교 전자계산공학과 교수

관심분야 : 소프트웨어공학, 분산객체컴퓨팅, 원격교육