XML 기반의 컴포넌트 명세 언어

°김원기*, 안치돈*, 이윤수**, 왕창종*
*[miso, joheunid]@selab.inha.ac.kr, cjwangse@inha.ac.kr, 인하대학교 전자계산공학과
**vslee@intra.ansantc.ac.kr. 안산공과대학 전산정보과

XML-Based Component Specification Language

° W. K. Kim *, C. D. Ahn*, Y. S. Lee**, C J. Wang*
*Dept of Computer Science & Engineering, Inha University
**Dept. of Computer Information, Ansan College of Technology

요약

컴포넌트 기반 소프트웨어 공학에서 컴포넌트를 명세하기 위한 명세 언어는 컴포넌트 분류, 검증 및 검색에 가장 기본적인 조건이다. 기존에 이미 많은 명세 언어가 사용되어 왔지만 명세의 복잡성으로 이한 어려움이 있다. 따라서 이 연구에서는 이러한 복잡성과 개발자에게 좀 더 쉬운 명세 방법을 제공하기 위해 XML 을 기반으로 하여 새로운 컴포넌트 명세 언어를 제안하였다.

제안한 명세 언어는 컴포넌트의 기능명세부분, 타입 검사에 의한 검색을 위한 부분, 명세 일치 방식을 위한 부분으로 구성된 컴포넌트 기능명세와 소프트웨어 아키텍처를 위한 명세로 구성된다. 이 연구에서는 각 부분을 XML 태그에 의해 정의하였다. 또한 소프트웨어 아키텍처 명세를 위한 그래픽 표기법과 텍스트 표기법을 제안하였다. 이 연구에서 제안한 명세언어는 화이트박스 재사용과 블랙박스 재사용을 지원하고 오류가능성을 줄일 수 있다.

1. 서론

세계적인 소프트웨어 시장에서 소프트웨어 개발 프로젝트에서의 주된 변화는 규모가 큰 소프트웨어 시스템을 개발, 분류, 갱신하는 방식으로 이루어지고 있다[1] 소프트웨어 시스템을 하나의 컴포넌트의 조합으로 설계하고 구현하는 것은 이제 더 이상 새로운 개념이 아니다 규모가 큰 소프트웨어 개발은 컴포넌트의 적절한 검색, 검증, 그리고 조합 과정을 통해 이룰 수 있다

컴포넌트 명세 언어(Component Specification Language)는 특정 문제에 대한 컴포넌트의 기능을 기술함으로써 컴포넌트 재사용성을 향상시키고 보다 쉽게 접근할 수 있는 방법을 제공한다 이를 위해서는 사용자의 요구 사항 정보를 추출하고 비교하는 과정이 필요하다.

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 이러한 한계를 해결하기 위해 문제 요구사항에 대한 설명, 컴포넌트의 기능, 그리고 컴포넌트 구조에 대한 명확한 정의를 제공하는 정형화된 명세 언어(Formal Specification Language)에 대한 연구가 현재 이루어지고 있다[2].

이 연구에서 제시하는 컴포넌트 명세언어는 컴포넌트 기능 명세와 소프트웨어 아키텍처 명세 두 형태의 명세언어를 제공한다 컴포넌트 기능 명세는 컴포넌트 기능을 명세함으로써 블랙박스 재사용(Black Box Reuse)을 지원하는 것이고 소프트웨어 아키텍처 명세는 구조적인 재사용 즉 화이트박스 재사용(White Box Reuse)을 지원한다 이 두 형태의 명세는 서로 개별적인 것이 아니라 컴포넌트 명세언어로 기술된 컴포넌트를 소프트웨어 아키텍처 명세에서 사용하는 방식을 취한다.

이 연구에서는 제안하는 명세언어는 XML(eXtensible Markup Language)을 기반으로 하고 있다 그리고 제안된 명세 언어를 XML 형태로 매핑 시킴으로 XML 의 기본적인 특징 중 하나인 구조적인 형태의 컴포넌트 명세 언어 형식을 나타낸다. 이는 개발자로 하여금 컴포넌트 명세 언어에 대한 특정 지식 없이 원하는 컴포넌트에 대한 요구 사항 정보를 기술할 수 있도록 한다 그리고 기술된 명세는 XML 문서로 저장되므로 데이터를 효율적으로 관리할 수 있다

2. 관련 연구

2.1 Rapide

Rapide 는 분산 시스템 구조의 아키텍처를 프로토타이핑(prototyping)하기 위해 설계된 이벤트 기반의 동시 객체 지향 언어(concurrent object-oriented language)이다.

Rapide 는 구현을 수행하기 전에 시스템 구조가 시뮬레이션을 위해 실행 가능한 형태로 표현될 수 있게 하는 아키텍처 구성을 제공하고, 분산 행위(behavior)와 타이밍(timing)을 정확하게 캡처(capture)하는 실행 모델(execution model)을 채택했다 또한 Rapide 는 참조 아키텍처(reference architectures)의 제약 조건 기반 정의(constraint-based definition)와 구조적 표준과의 일치를 위한 시스템 테스팅을 제공하기 위한 정형화된 제약 조건과 매핑(mapping)을 제공한다[3].

Rapide 에서 아키텍처는 몇 가지 요소에 의해 구성된다 이 구성 요소는 모듈에 대한 명세 집합, 인터페이스간 직접적인 통신(communication)을 정의하는 연결(connection) 규칙의 집합, 그리고 통신 패턴을 정의하는 정형화된 제약 조건이다. 인터페이스는 다른 모듈 인터페이스에 제공되는 특징들에 대해 정의한 것으로, 모듈의 행위에 대한 추상적인 정의 또한 가능하다. 따라서 인터페이스 행위는 모듈에 의해 수신된 데이터와 생성된 데이터간의 관계를 명세하는 것이다 그리고 연결은 인터페이스간 데이터에 대한 동기적, 또는 비동기적 통신을 정의하는 것으로, 이벤트 패턴

매칭을 사용하여 정적, 그리고 동적 아키텍처 정의를 가능하게 한다. 정형화된 제약 조건을 기술한다는 것은 인터페이스와 연결에 대한 제약 사항을 명세하는 것을 말한다

아키텍처 생성(architecture constructs)은 이벤트간 의존성을 표현하는 poset(partially ordered event set) 형태의 이벤트 기반 실행 모델(event-based execution model)을 가진다. Rapide 에서 아키텍처 정의는 모듈을 기술하기 전에 이루어지고, 이는 모듈들을 결합하기 위한 프레임워크이며, 모듈간 통신을 제약한다

각각의 모듈과 인터페이스는 독립적으로 그리고 동시적으로 수행될 수 있다. 각각의 인터페이스에서의 이벤트는 인터페이스 제약조건을 만족해야 하고, 모든 인터페이스에서의 이벤트는 아키텍처의 제약조건을 만족해야 한다.[4]

Rapide 는 크게 다섯 개의 기술언어로 분류된다 컴포넌트 인터페이스를 기술하기 위한 타입 언어(type language), 컴포넌트간 이벤트의 흐름을 기술하기 위한 아키텍처 언어(architecture language), 컴포넌트 행위에 추상적인 제약조건을 기술하기 위한 명세 언어(specification language), 실행 가능한 모듈을 기술하기 위한 실행 언어(executable language), 그리고 이벤트 패턴을 기술하기 위한 패턴 언어(pattern language)가 있다.

Rapide 는 poset 을 통한 시뮬레이션 및 아키텍처 행위에 대한 분석을 지원하고, 기술된 시스템 아키텍처를 이용한 교착 상태(deadlock)나 제약 조건의 위배 등에 관한 분석이 가능하다.

2.2 C2

C2 는 메시지 기반 스타일(message-based style)을 사용하여 사용자 인터페이스를 기술하는 언어이다. 이를 위해 일반적인 컴포넌트와 연결자(connector)의 구성에 대한 제약 조건과 연결 규칙을 정의해야 한다 C2 에서의 연결자의 역할은 컴포넌트에서 발생한 이벤트를 다른 컴포넌트에 전달하는 것인데, 제약 조건으로 아래 방향으로는 통지(notification) 이벤트만을, 윗 방향으로는 요구(request) 이벤트만을 전달한다는 방향성에 제한을 두고 있다[5]

현재 C2 는 시스템의 아키텍처를 기술할 수 있는 도구와 기술된 모델의 정확성을 검사하는 등 다양한 기능을 제공하는 도구가 개발되어 지원되고 있으며, C2 를 이용하여 아키텍처 연결자를 구현할 때 아키텍처와 미들웨어(middleware) 기술의 관계에 대한 연구가 이루어지고 있다 [6]

3. XML 기반 컴포넌트 기능 명세 언어

이 장에서는 제안한 XML 기반의 컴포넌트 명세 언어를 정의하기 위해 필요한 요소들에 대해 살펴보고, 요소들을 DTD 로 정의한다. 정의한 DTD 에 의해서 XML 기반의 명세 언어 구조를 정의하며, 이 연구에서 제시하는 컴포넌트 기능 명세언어의 특징을 살펴본다.

3.1 컴포넌트 기능 명세를 위한 요소

컴포넌트 명세(Component Specification)를 위한 명세 언어를 정의하기 위해서는 가장 먼저 명세를 위해 필요한 요소가 무엇이 있는지에 대한 것이다. 이 연구에서 제안한 XML 기반 컴포넌트 명세 언어의 정의에 필요한 요소를 표 1 에 나타내었다. 이러한 요소들은 XML 에서 각각의 태그로 표현된다.

표 1 컴포넌트 명세 언어 정의에 필요한 요소

요 소	의 미
component_name	정의된 컴포넌트 이름
Description	컴포넌트 기능을 자연어로 기술
Uses	컴포넌트에서 사용되는 추상형 데이터 타입
sig_name	컴포넌트 내의 메소드명
Direction	입출력에 대한 방향성
Param_name	선언된 파라미터 이름
Param_type	메소드의 파라미터 타입
behavior_name	컴포넌트의 기능을 수행하는 메소드명
Modifies	메소드 수행시 값이 변하는 변수명
Ensures	메소드 기능의 대수적 표현

3.2 XML 기반 명세 언어를 위한 DTD 생성

XML 에 기반하여 컴포넌트를 명세하기 위해서는 명세에 필요한 요소들에 대한 정의가 필요하다 이는 XML 에서 사용되는 사용자 태그가 유효하도록 하고 각각에 대한 데이터 타입을 정의함으로써 사용자의 잘못된 데이터 타입 입력에 대해 검사할 수 있는 기본을 제공한다.

이 연구에서 사용된 기본적인 요소들에 대한 정의는 그림 1 과 같은 형식으로 나타난다.

```
<?xml version="1.0"?>
<!ELEMENT component_architecture (components)>
<!ELEMENT components (component_name, description,
 uses*, signature*, behavior+)>
<!ELEMENT signature (require*)>
<!ELEMENT require (name)>
<!ELEMENT name (direction, param_name, param_type)>
<!ELEMENT behavior (names+, modifies*, ensures+)>
<!ELEMENT component_name (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT uses (#PCDATA)>
<!ELEMENT sig_name (#PCDATA)>
<!ELEMENT direction (#PCDATA)>
<!ELEMENT param_name (#PCDATA)>
<!ELEMENT param_type (#PCDATA)>
<!ELEMENT behavior_name (#PCDATA)>
<!ELEMENT modifies (#PCDATA)>
<!ELEMENT ensures (#PCDATA)>
```

그림 1 XML 기반 명세 언어를 위한 DTD

3.3 XML 기반의 명세 언어 구조

정의한 DTD 를 기본으로 한 컴포넌트 명세 언어 기본구조는 그림 2 와 같은 형태를 가진다. <component_architecture> 태그를 최상위 태그로 나타냈으며, 각각의 하위 태그들을 계층적으로 표현함으로써 개발자가 그 의미를 파악하고 데이터를 입력하는데 어려움이 없도록 표현하였다. 기존 명세 언어의 복잡성을 피하고 XML 의 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 일관성 또한 그대로 유지할 수 있고, 기본적으로 제공되는 뷰와 DTD 를 통해 오류 발생률 또한 감소시킬 수 있다 이 연구에서 사용한 기능 명세는 행위일치 검색을 지원하기 위해 자바 언어 형태를 따라 정의하였다.

```
<?xml version="1.0"?>
<!DOCTYPE component_architecture PUBLIC
 "http://selab.inha.ac.kr/dtd/component_arch.dtd">

<component_architecture>
<components>
   <component_name> </component_name>
   <description> </description>
   <uses> </uses>
   <signature>
      <require>
         <sig_name>
            <direction> </direction>
         <param_name> </param_name>
```

```
            <param_type> </param_type>
        </sig_name>
      </require>
    </signature>
    <behavior>
      <behavior_name> </behavior_name>
      <modifies> </modifies>
      <ensures> </ensures>
    </behavior>
  </components>
  </component_architecture>
```

그림 2 XML 기반 컴포넌트 명세 언어 구조

4. 소프트웨어 아키텍처 명세 언어

이 장에서는 소프트웨어 기능 명세를 위해 필요한 요소와 이를 위해 필요한 그래픽 표기법과 이 표기법의 XML 매핑에 관하여 살펴본다

4.1.소프트웨어 아키텍처 명세를 위한 요소

소프트웨어 아키텍처는 전체 시스템의 컴포넌트와 컴포넌트 사이의 상호작용, 연결을 관할하는 커넥터를 중심으로 분석하고 기술함으로써 시스템의 구조를 명세 한다. 이 연구는 이러한 소프트웨어 아키텍처도 하나의 컴포넌트라는 관점으로 명세에 접근하고 메시지 기반의 아키덱처 명세를 제안한다.

4.2 그래픽 표기법

그림 3 은 이 연구에서 제안하는 그래픽 표기법을 이용한 아키텍처 명세의 예이다.

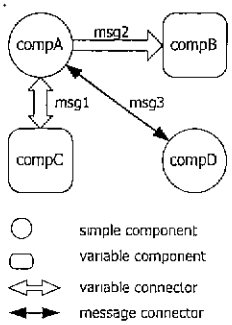

그림 3. 그래픽 표기법의 예

단순 컴포넌트(simple component)는 일반적인 컴포넌트를 표현하기위한 표기법이고 가변 컴포넌트(variable) 컴포넌트는 컴포넌트의 형이 결정되지 않은 것을 표현하기 위한 표기법이다 변동 컴포넌트는 화이트박스 재사용을 위한 표기법으로 UNIX 의 파이프라인과 같은 구조를 표현하는데 사용한다.

메시지의 교환을 통해 컴포넌트간의 통신이 이루어지는 소프트웨어 아키텍처에서 발생할 수 있는 메시지의 형태는 단방향과 양방향이 있다 가변 커넥터는 가변 컴포넌트와 함께 화이트박스 재사용을 위한 표기법이다

4.3 XML 기반의 텍스트 표기법

그림 4 는 그림 3 의 그래픽 표기법중 compA 부분을 텍스트 표기법으로 변환한 예이다.

```
<composite_component>
<simple_component>
<name> compA </name>
<in_message>
<var_msg>
<src> compC </src> <msg_name> msg1 </msg_name>
</var_msg>
<simp_msg>
<src> compD </src> <msg_name> msg3 </msg_name>
</simp_msg>
</in_message>
<out_message>
<var_msg>
<dest> compB </dest> <msg_name> msg2 </msg_name>
<dest> compC </dest> <msg_name> msg1 </msg_name>
</var_msg>
<simp_msg>
<dest> compD </dest> <msg_name> msg3 </msg_name>
</simp_msg>
</out_message>
</simple_component>
</composite_component>
```

그림 3. 소프트웨어 아키텍처 텍스트 표기법

5. 결론

이 연구에서는 소프트웨어 컴포넌트를 명세하기 위한 XML 기반의 명세 언어를 제안하였다 구현을 고려한 정형화된 명세 언어의 복잡성을 줄이고, 개발자에게 쉬운 접근 방법을 제공하기 위해 특정 언어형태의 기능 명세를 제공했다

컴포넌트 명세를 XML 로 정의함으로써 컴포넌트 명세 요소들은 계층적인 구조를 가지며, 개발자는 뷰를 통해 이 구조를 볼 수 있다. XML 문서의 유효성은 DTD 에 사용된 태그들과 데이터 타입을 정의함으로 유지된다. 그리고 DTD 에 정의된 데이터 타입을 벗어나는 잘못된 입력에 대한 검사는 개발자의 오류 발생 가능성을 감소시킨다.

소프트웨어 아키덱처 명세를 위한 표기법을 제공함으로써 컴포넌트의 블랙박스 재사용뿐만 아니라 아키텍처의 구조적 재사용 즉 화이트 박스 재사용을 제공했다

향후 연구과제로는 컴포넌트 기능 명세와 소프트웨어 아키텍처 명세를 통합한 명세로 소프트웨어를 하나의 컴포넌트로서 완전히 기술할 수 있는 명세가 연구되어야 할 것이다.

참고 문헌

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

[2] Jun-Jang Jeng and Betty H. C. Cheng, "A Formal Approach to Reusing More General Components," *IEEE Proceedings of 9th Knowledge-Based Software Engineering Conference*, Monterey, California, pp. 90-97, September 1994.

[3] D. C. Luckham, J. Kenney, L. M. Augustin, J. Vera and D. Bryan, W. Mann, "Specification and Analysis of System Architecture Using Rapide," *IEEE Transaction on Software Engineering*, Vol. 21, No. 4, 1995.

[4] D. C. Luckham and J. Vera " An Event-Based Archtecture Definition Language", *Proceedings of the International Conference on Software Engineering*, 1999.

[5] P. Oreizy, N. Medvidovic and R. N. Taylor, "Architecture-Based Runtime Software Evolution," *Proceedings of the International Conference on Software Engineering*, 1998.

[6] E.D. Nitto and D.Rosenblum, "Exploiting ADLs to Specify Architectural Styles Induced by Middleware Infrastructures" *Proceedings of the International Conference on Software Engineering*, 1999.