자바 빈즈 컴포넌트의 내부 종속성 관계를 고려한 XML 기반 명세 기법에 관한 연구

o 김종선,심우곤,류기열,백인섭
아주대학교 정보통신공학과
{netmania,startear,kryu,ispaik}@madang.ajou.ac.kr

A Study on XML based JavaBeans Component Specification with Considering the Intra-component Dependency

o Jong-Seon Kim, Woo-Gon Shim, Ki-Yeol Ryu, In-Sub Paik
*Graduate School of Information and Communication, Ajou University.

요 약

컴포넌트에 기반한 소프트웨어 개발(CBSD)에 관한 연구가 활발히 진행 중에 있다. 시스템 개발 시 컴포넌트를 활용하여 개발기간과 신뢰도를 향상시키기 위해서는 무엇보다 이들 컴포넌트의 탐색과 선택을 효과적으로 지원할 수 있는 방안이 요구된다[11]. 이를 위하여 컴포넌트 명세기법과 검색시스템에 관한 연구가 지속적으로 수행되고 있으나, 다수의 연구에서 컴포넌트의 기능과 인터페이스에만 초점을 두어 종속성에 대한 고려는 부족한 실정이다. 컴포넌트는 본질적으로 완벽하게 독립적이지 못하며 내부와 외부 요소간 종속성을 맺기 때문에 [5], 종속성에 관한 명세와 정보제공은 컴포넌트 재사용과 활성화를 위해 필수적이다. 따라서, 본 연구에서는 XML 을 이용한 자바빈즈 컴포넌트의 내부 종속성 명세기법을 제시하고자 한다.

1. 서 론

소프트웨어의 생산성 향상을 위한 방법으로 소프트웨어 재사용(reuse) 중의 컴포넌트 기반 소프트웨어 개발(CBSD: Component-Based Software Development) 방법론이 꾸준하게 성장하고 있다.

컴포넌트 기반의 소프트웨어를 개발하기 위해서 외부로부터 컴포넌트를 구입하는 방법은 아키텍처 불일치(architectural mismatch)[6]의 문제를 가지고 있다. 소프트웨어 아키텍처란 소프트웨어 시스템을 구성하는 컴포넌트들의 구조와 상호 관계를 설명하는 시스템에 대한 추상화(abstraction)이다. 아키텍처 불일치의 문제는 일반적으로 컴포넌트와 컴포넌트 기반 소프트웨어를 구축시, 각각의 생산자와 소비자는 서로 다른 조직체로 상이한 아키텍처를 기반 하기 때문에 발생한다. 이러한 아키텍처 불일치를 간과하고 단지 기능적 적합성에 따라 컴포넌트를 선택하는 경우 전체 시스템은 올바르게 작동하지 않을 수 있다. 컴포넌트는 내부와 외부의 종속성을 가지는데 이들 종속성을 파악하는 것은 하나의 컴포넌트로 인해 영향을 받을 수 있는 잠재적인 영향력을 파악하는 것으로 시스템의 유지 보수 및 통합에 크게 기여한다. 외부 종속성을 파악하는 것은 인터페이스간의 연결을 통해 확인해 볼 수 있으나 내부 종속성은 쉽게 확인할 수 없다. 하지만 내부 종속성을 파악하여 컴포넌트 내부의 구현된 메소드등의 요소들간의 링크를 분석하는 것은 유지보수 및 재사용성을 확인하는데 효과적으로 작용한다.

XML 은 의미 정보를 기술할 수 있고 문서 내부의 논리적 구조와 플랫폼에 독립적인 특성을 제공한다[7]. 때문에 컴포넌트의 종속성을 표현하는데 필요한 정보들을 기술하는데 매우 적합하다.

본 논문에서는 자바 빈즈(JavaBeans)를 컴포넌트 모델로 하여 이를 컴포넌트의 내부 종속성 관계를 파악하여 XML 기반의 명세 기법을 제안한다.

2. 기존 연구

본 장에서는 논문과 관련된 일반적인 연구로 자바 빈즈 컴포넌트의 명세와 컴포넌트의 종속성에 관련된 내용을 소개한다.

2.1 XML 기반의 자바 빈즈 컴포넌트의 명세 방법

다음의 DTD 는 자바 빈즈 컴포넌트를 명세한 기존 연구의 결과물이다[13].

```
<!ELEMENT document(bean)>
<!ELEMENT bean
(className,superClasses,interfaces,properties,m
ethods,events)>
<!ELEMENT className(#PCDATA)>
<!ELEMENT superClasses(superClass)*>
<!ELEMENT superClass(#PCDATA)>
<!ELEMENT interfaces(interface)*>
<!ELEMENT interface(#PCDATA)>
<!ELEMENT properties (property)*>
<!ELEMENT property(propertyType, propertyName,
readMethod, writeMethod)>
<!ELEMENT propertyType (#PCDATA)>
<!ELEMENT propertyName (#PCDATA)>
<!ELEMENT readMethod (#PCDATA)>
<!ELEMENT writeMethod (#PCDATA)>
<!ELEMENT methods (method)*>
<!ELEMENT method (#PCDATA)>
<!ELEMENT events (event)*>
<!ELEMENT event
(listenerType,listenerMethods,addListenMethod,r
emoveListenerMethod)>
<!ELEMENT listenerType (#PCDATA)>
<!ELEMENT listenerMethods (#PCDATA)>
<!ELEMENT addListenMethod (#PCDATA)>
<!ELEMENT removeListenerMethod (#PCDATA)>
```

그림 1 자바 빈즈 컴포넌트 명세를 위한 DTD

그림 1 의 컴포넌트 명세 방법에 있어서 가장 큰 장점은 명세에 필요한 최소한의 정보를 포함하도록 정의하여 간단하게 사용할 수 있다는 점이다. 자바 빈즈에서는 내부 투영(Introspection)

을 통해서 이러한 명세에 필요한 정보를 추출하도록 제공한다[8]. 제안된 DTD(Document Type Definition)는 이 내부 투영을 이용하여 프로퍼티(Property), 메소드(Method), 이벤트(Event)에 대한 구체적인 정보를 얻을 수 있었다.

기존의 연구에서 제안된 DTD는 내부 종속성을 표현하고자 하는 자바 빈즈 컴포넌트의 명세로는 적합하지 않다. 프로퍼티, 메소드, 이벤트요소를 포함하는 인터페이스를 하나의 엘리먼트로 선언한 것이나 메소드를 단순화한것등이 그것이다. 이를 해결하기 위해서 프로퍼티나 메소드를 표현한 명세 이외에도 컴포넌트가 가지고 있는 종속성을 표현한 명세의 추가가 필요하다. 본 논문에서는 이러한 내부 종속성을 표현하기 위해서 DTD를 재정의하여 제안하였다.

2.2 아키텍쳐 수준의 컴포넌트 종속성 분석 기법

전통적인 종속성 분석 기법은 ①프로그램 코드 분석을 바탕으로 ②데이터의 흐름(data flow)을 제어하고 ③전체 시스템을 통제(control)하는 방법이었다. 이 방법을 통해서 다음과 문제들을 해결할 수 있다.

(1) 시스템의 어느 구성 요소가 어떠한 이벤트를 발생하는가
(2) 시스템의 오류가 발생했을 시에 어느 컴포넌트를 교체하여 문제를 해결할 수 있는가
(3) 컴포넌트는 교체했을 경우에 문제가 발생될 수 있는 종속적인 구성 요소는 무엇인가

이러한 문제를 해결하기 위해서 전제되는 것은 전체 시스템의 프로그램 코드 분석이다. 때문에 개발 이전 단계의 종속성을 파악하는 일이 매우 힘들어진다.

한편, 아키텍쳐 수준에서 종속성 관계를 파악하고 있다면 전체 프로그램 코드를 분석할 수 없는 개발 초기라도 위에서 언급했던 문제들을 해결할 수 있다.[9]. 체인 기법(chaining)은 아키텍쳐 수준에서 ADL 에 기반하여 종속성을 분석하는 기법으로 제안되었으며 다음과 같은 종속성을 확인한다[10].

(1) 컴포넌트 결합에 의한 외부 종속성:
다른 클래스의 인스턴스를 생성하여 사용하거나(Aggregation) 이벤트 발생시 서비스를 제공하는 다른 컴포넌트에게 처리를 요청하는(Delegation)과 같은 행동이 표현될 수 있다.

(2) 컴포넌트 내부의 요소들간의 내부 종속성:
동일 클래스 내의 함수 호출이나 데이터 조작과 같은 행동을 통해서 나타날 수 있다. 인터페이스와 같은 in(required), out(provided)의 두개의 port 를 통해서 in port 에서 받은 서비스를 컴포넌트안의 메소드를 이용해서 처리하고 이를 out port 로 넘겨주는 것으로 동작이 이루어진다.

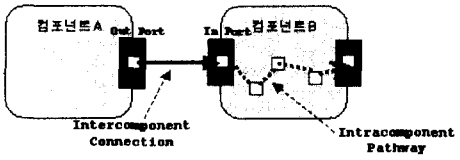

그림 2 컴포넌트 사이의 종속성을 가질수 있는 유형

대다수의 연구가 컴포넌트 간의 연동에만 초점을 맞춘데 반해 체인 기법은 내부 종속성을 분석함으로서 구체적인 Impact(변경 혹은 교체 시 받는 영향)영역파악 및 테스트 컴포넌트의 도출을 지원한다.

2.3 컴포넌트의 결합방법

컴포넌트들을 쉽게 결합하거나 혹은 개발된 후에도 다른 부품 컴포넌트로의 교체가 용이 하려면 컴포넌들간의 결합 방법과 그 성질들을 파악해야 한다. 컴포넌트를 결합하는 방법은 다음과 같은 주요 세 가지의 방법을 통해서 이루어지는데[12] 본 논문에서는 이를 이용한다.

- 사용 관계(use relation)에 의한 결합
- 상속 관계(inheritance relation)에 의한 결합
- 병렬 관계(parallel relation)에 의한 결합

그림 3 에서 보여 주는 것처럼 사용 관계에 의한 결합은 하나의 컴포넌트의 제공 인터페이스에서 제공하는 서비스를 다른 컴포넌트가 사용할 때 가능한 결합 방법이다. 상속 관계에 의한 결합은 두 컴포넌트의 요구 또는 제공 인터페이스간의 상속이 일어날 때 가능한 결합 방법으로 하나의 인터페이스가 다른 하나의 인터페이스의 서브 클래스로 된다. 병렬 관계에 의한 결합은 두 개의 컴포넌트가 사용 관계를 가지고 있지는 않지만 하나의 시스템을 구성하기 위해서 두개의 컴포넌트를 모아 놓은 것이다.

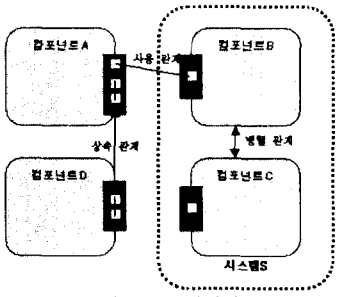

그림 3 컴포넌트 결합의 종류

3. 자바 빈즈의 종속성 분석 및 표현 기법

본 장에서는 관련 연구들을 통한 내용을 바탕으로 자바 빈즈 컴포넌트의 내부 종속성에 대해서 표현한다. 이로써 앞서 밝힌 장점들을 수용함과 동시에 구현의 용이성을 모색한다.

3.1 제안을 위한 가정

본 논문에서 제안하는 컴포넌트의 종속성은 다음의 사항을 가정한다.

1. 자바 빈즈 컴포넌트는 그래픽 인터페이스(UIE: user *interface element*)를 구현한 가시적인 것과 그렇지 않은 기능만을 담당한 비가시적인 것으로 구분되는데 본 논문은 기능만을 구현한 컴포넌트를 고려하여 진행한다. 자바 빈즈 컴포넌트중에는 그래픽 인터페이스가 가미된 것이 상당수 존재하지만 로직(logic)만을 구현한 자바 빈즈 컴포넌트도 많이 존재하고 특히 서버 사이드에서는 비가시적인 컴포넌트가 선호되고 있는 실정이다. 추가적으로 이는 본 논문의 구현상 컴포넌트 내부 요소들에 대한 분석 및 표현을 위한 목적으로 제안되었다.
2. 컴포넌트에서 사용된 메소드 중에서 public 메소드만을 고려한다. 이는 CORBA 의 IDL 과 같은 방법으로 인터페이스에서 자바 빈즈의 서비스를 호출하는데 사용되는 메소드는 public 이기 때문이다[3].
3. 컴포넌트의 결합방법중에서 상속관계와 병렬관계를 제외한 사용관계만을 기술하도록 한다. 병렬관계는 사실상 사용관계를 가지고 있지 않고 구성되어 있기 때문에 내부종속성을 가지고 있다고 볼수 없다. 상속관계의 종속성은 상속관계의 부모 컴포넌트가 가진 속성에 대해 분석하고 표현하기가 쉽지 않았다.

3.2 분석 및 표현 기법

컴포넌트 내부의 종속성을 표현하기 위한 자바 빈즈 컴포넌트 LoginProcess 는 다음과 같은 내부 요소가 있다고 가정한다 [그림 4].

```
1  public class LoginProcess {
2    private String  id     = null;
3    private String  pass   = null;
4    private boolean fAuthed = false;
5
6    public void    setId   (String id   )  { this.id = id; }
7    public void    setPass (String passwd) { pass = passwd; }
8    public String  getId   ()              { return id;     }
9    public boolean IsAuthed()              { return fAuthed;}
10   public boolean check  () {
11     ....
12     // JDBC를 사용자 데이타베이스에 접속함
13     query = "SELECT * FROM USERS WHERE ID='" + id +
14             "' AND PASSWORD='" + pass + "'";
15     ....
16     if(rs.next()) {
17       fAuthed = true;
18       return fAuthed;
19     }
20     return fAuthed;
21   }
22 }
```

그림 4 자바빈즈 컴포넌트 예제

종속성 확인을 위한 시스템에서는 이 정보와 메소드를 파싱(parsing)하여 종속성 확인 테이블(dependency table)을 생성한다. 종속성 확인 테이블은 자바 빈즈 컴포넌트에서 사용한 메소드와 포트 타입, 모든 프로퍼티의 이름으로 이루어진다. 또한 메소드를 파싱하여 메소드가 사용한 모든 프로퍼티를 체크 한다. 그림 5는 위의 자바빈즈 예제를 분석한 종속성 확인 테이블이다.

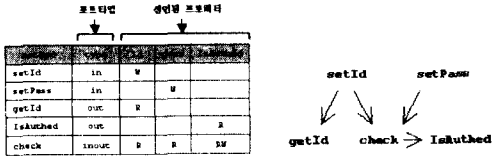

그림 5 종속성 확인 테이블(Dependency Table)

메소드의 내부에 선언되어 일시적으로 사용되는 변수들은 종속성 체크에서 제외되었다. 메소드들 간의 종속성을 파악하는 데에는 무관하기 때문이다. 이렇게 분석된 종속성 정보를 기술하기 위한 DTD 과 XML 문서의 예는 그림 6 과 같다. 이는 기존 연구[13]에서 제안한 DTD 를 확장한 것으로 내부 종속성을 표현한 것 이외에도 인터페이스와 메소드 정의를 아무런 계층 구조 없이 같은 레벨의 엘리먼트로 표기하는 등의 오류를 수정하였다.

4. 결론 및 향후 연구과제

본 연구에서는 자바 빈즈 컴포넌트의 내부 종속성 확인을 위한 기법을 제시하였다. 자바 빈즈 컴포넌트에서 종속성 관계를 도출하는 방법은 프로그래머에 의한 정형화된 기술을 통해서 표기하여 가능하였고 이를 바탕으로 컴포넌트의 XML 기반의 명세를 작성하였다. 이를 통해 컴포넌트의 재사용과 컴포넌트 기반 소프트웨어 개발 후의 유지 보수에 안정을 도모하고 이를 지원할 수 있다.

향후, 본 논문에서 프로퍼티에 기반한 종속성외에도 메소드의 호출간에 발생하는 종속성을 분석할 필요가 있다. 뿐만 아니라 제외되었던 상속 관계에 의한 컴포넌트 결합에서의 종속성 확인과 그래픽 인터페이스를 기반한 자바 빈즈에서의 정보 추출의 부분이 추가될 것이다.

```
<!ELEMENT Methods(Method)*)>
  <!ELEMENT Method(MethodName, ArgName*, DependMethodName*)>
    <!ELEMENT MethodName (#PCDATA)>
    <!ATTLIST MethodName
      returnType    CDATA    #REQUIRED
    >
    <!ELEMENT ArgName (#PCDATA)>
    <!ATTLIST ArgName
       ArgType          CDATA    #IMPLIED
    >
    <!ELEMENT DependMethodName (#PCDATA)>
```

```
...
<Method>
<MethodName retutunType=void>setId</MethodName>
<ArgsName argType=String>id</ArgsName>
</Method>
<Method>
<MethodName retutunType=void>setPass</MethodName>
<ArgsName argType=String>passwd</ArgsName>
</Method>
<Method>
<MethodName retutunType=String>getId</MethodName>
<DependMethodName>setId</DependMethodName>
</Method>
<Method>
<MethodName retutunType=boolean>IsAuthed</MethodName>
<DependMethodName>check</DependMethodName>
</Method>
<Method>
<MethodName retutunType=boolean>check</MethodName>
<DependMethodName>setPass</DependMethodName>
<DependMethodName>setId</DependMethodName>
</Method>
...
```

그림 6 수정된 DTD 와 XML 문서

참고문헌

[1] A.Beugnard et al., "Making Components Contract Aware", IEEE Computer, 32(7), pp.38~45, July 1999.

[2] B.Felix, et al. , "Volumell : Technical Concepts of Component Based Software Engineering", Carnegie Mellon Software Engineering Institute, CMU/SEI-2000-TR-008, 2000.

[3] CORBA 2.4.2 specification, Object Management Group, 2001.

[4] D.C. Luckham, J.J. Kenney, L.M. Augustin, J. Vera, D. Bryan, and W. Mann, "*Specification and Analysis of System Architecture Using Rapide*", IEEE Transactions on Software Engineering, 21(4):336-355, April 1995.

[5] D. E. Perry, "System Compositions and Shared Dependencies," 6th Workshop on Software Configuration Management, ICSE 18, Berlin Germany, March 1996.

[6] D.Garlan., R.Allen, J.Ockerbloom, " Architectural Mismatch: Why Reuse is So Hard", IEEE Software 12, 6 (Nov.), 17—26, 1995.

[7] Extensible Markup Language (XML) 1.0 (Second Edition),W3C, 2000.

[8] G. Hamilton, "JavaBeans Specification Version 1.01", Sun Microsystems, 1997.

[9] J. A. Stafford and A. L. Wolf, " Architecture-Level Dependence Analysis for Software Systems", University of Colorado, 2000.

[10] J.A.Stafford, D.J.Richardson, A.L.Wolf, "Aladdin: A Tool for Architecture-Level Dependence Analysis of Software Systems", University of Colorado, 1998.

[11] J. Kontio, G. Caldiera and V. R. Basili, "Defining Factors, Goals and Criteria for Reusable Component Evaluation," in Proc. of CASCON 96, Toronto, Canada, Nov. 12-14, 1996.

[12] 류기열, 이정태, 김윤명 "유연한 결합을 지원하는 컴포넌트 모델" , 1999 년 동계 워크샵, 한국정보과학회 프로그래밍언어 연구회, 1(1), pp.59-72, 1999.

[13] 이성은, 김영익, 류성열 " 자바 프로그램의 재사용을 위한 자바 빈즈 컴포넌트의 추출 및 명세화 기법" , 정보처리학회논문지 VOL. 7 NO. 5 pp. 1388-1400, 2000.