컴포넌트 의존정보 명세화와 의존 관리자의 구현

최 상 균[†] · 송 영 재[††]

요 약

컴포넌트 기반 소프트웨어 개발(Component-based software Development) 환경에서 컴포넌트간의 의존관계 정보를 제공하는 것은 컴포넌트를 합성하는 개발자에게 중요한 일이다. 그러나 컴포넌트의 의존관계 정보가 충분하게 제공되지 않고 있다. 따라서 개발자는 원시 컴포넌트의 의존관계를 모르고 컴포넌트를 합성하는 경우가 발생한다. 본 논문에서는, 위와 같은 단점을 보완하기 위하여, 컴포넌트 사이의 의존정보를 XML로 정의하여 관리하고, 개발자에게 원시 컴포넌트 의존관계 정보를 제공하여, 무결성 있고 영속성 있는 정보 시스템 개발을 위한 컴포넌트 의존정보를 제공하는 의존 관리자의 설계와 구현을 하였다. 의존 관리자에서 생성된 컴포넌트는 재사용성 측정 메트릭의 측정값이 89%에서 95%가 신뢰구간 안에 포함되어 있음을 보이고 있다.

키워드 : 컴포넌트 형상관리, 의존관리

The specification of component dependence information and implementation of dependence manager

Sang-kyoon Choi[†] · Young-Jae Song[††]

ABSTRACT

In the component-based software development environment, it is important for developers who compose components to provide the information of dependence relationship between components. However it is not provided the information of dependence relationship of components. Therefore, it happens that the developer occasionally composes the components without knowing the dependence relation between the primitive components. In order to complement these disadvantages, In this paper, the design and implementation of dependence manager which provides the component dependence information for the development of integrity and persistence information system are established by defining the information about the dependence and relation between components as XML, managing them and providing a developer with the information about the dependence relationship between primitive components, we can establish. The components being created in the dependence manager show that the figure of reused matrix measurement is indicated from 89% to 95%, being included in confidence limits.

Key Words : Component Configuration Management, Dependence Management

1. 서 론

컴포넌트 기반 소프트웨어 개발(Component-based software Development)이 소프트웨어 개발에 있어 새로운 패러다임으로 인식되고 있다. 그러나 CBD에 관한 형상관리 연구가 뒤따르지 못하고 있는 실정이다[1]. CBD와 전통적인 소프트웨어 개발 방법에 근거하여 CBD의 형상관리 요구사항을 보면, 컴포넌트의 버전관리와 무결성 유지로 이는 일관성 체크와 형상 항목간의 의존성을 관리하는 것으로 구분된다[2]. 컴포넌트 형상관리를 위해서는 컴포넌트간의 의존관리가 필수적인 행위이다. 컴포넌트 합성에서 컴포넌트가 새롭게 추가될 때, 물리적인 이진 컴포넌트가 전달되지 않기 때문에 컴포넌트 사이의 의존관계 정보는 전달되지 않는다. 따라서 이 컴포넌트가 시스템에 어떤 영향을 미치는지에 대한 정보를 알아야 한다. 의존관리는 시스템 전반에 걸친 모든 컴포넌트에 관련된 정보를 메타 데이터에 의한 의존 그래프 인터페이스를 통해서 가능하다. 메타 데이터는 새로운 인터페이스로 컴포넌트 등록을 할 때 저장소에 저장된다. 컴포넌트 저장소는 비즈니스 어플리케이션 구축을 위해 요구되는 컴포넌트를 찾고 공급하는 어플리케이션 개발을 위한 환경으로서, 컴포넌트의 개발과 유통 그리고 활용을 지원하기 위한 실질적인 도구가 되는 라이브러리 시스템이다[3]. 이를 활용하여 컴포넌트를 합성할 경우, 개발자는 이미 생성된 원시 컴포넌트의 의존관계를 알아야 하며, 개발자는 이를

† 정 회 원 : 김포대학 컴퓨터계열 교수
†† 종신회원 : 경희대학교 전자정보학부 교수
논문접수 : 2005년 1월 25일, 심사완료 : 2005년 4월 22일

이용하여 컴포넌트 의존관계를 분석하고 개발 작업을 하여야 한다. 그러나 기존의 구축된 컴포넌트 저장소에서는 컴포넌트 메타 데이터 등의 정보만을 제공하고 있다. 컴포넌트의 의존관계 정보를 모르면, 개발자는 향후 원시 컴포넌트의 의존 관계가 변경될 경우 발생할 문제점들을 예측하지 못하는 상황에서 시스템 개발과 유지보수 작업을 하게 된다. 특히, 분산된 개발환경에서의 컴포넌트 합성은 향후 발생할 수 있는 여러 가지 문제점을 내포할 수 있다.

본 논문에서는 위와 같은 단점을 보완하기 위해서, 컴포넌트 사이의 의존 정보를 XML로 명세화 하여 컴포넌트 사이의 의존 정보를 표현하였고, 컴포넌트를 합성하는 과정에서 의존정보를 관리하는 의존 관리자의 설계 및 구현을 하였다. 본 논문은 2장에서 컴포넌트 의존에 관한 관련연구에 대하여 고찰하고, 3장에서는 의존관계 명세를 제시하였고, 4장은 컴포넌트의 의존성을 분석하였으며, 5장은 의존 관리자의 설계와 구현을, 6장은 평가를, 7장은 결론 및 향후 연구방향을 정리하였다.

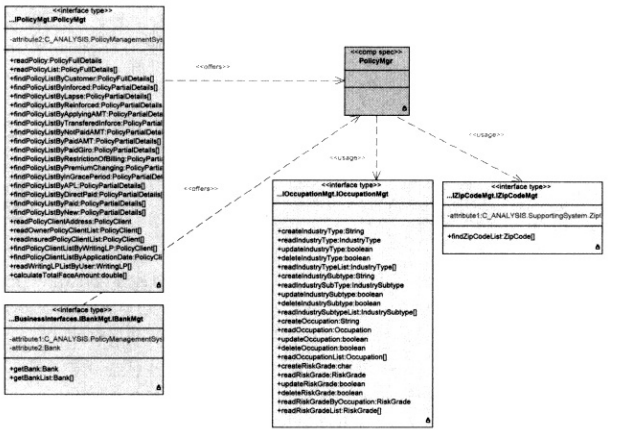
2. 관련 연구

2.1 컴포넌트 의존관리

컴포넌트 구조 및 의존관리는 컴포넌트 형상관리의 핵심 부분이다[4]. 컴포넌트가 조립되고 실행시간에서 관리될 때, 컴포넌트간의 의존관계를 알아내는 것은 중요하다. 개발자는 개발단계에서 컴포넌트간의 의존관리부터 분석하고 개발을 시작하여야 한다. 현재의 대표적인 컴포넌트 모델인 COM, EJB, CCM은 컴포넌트 사이의 명시적인 의존 명세를 요구하지 않고, 의존을 관리하지도 않는다[5]. 그러나 컴포넌트 사이의 의존이 명시적으로 명세화되지 않으면, 강력한 컴포넌트 기반의 시스템을 구축하는데 어려움이 발생할 것이다. 특히 동적인 재구성 시스템은 더욱 어려울 것이다. [5]에서는 정적 및 동적 의존관리로 구분하여 컴포넌트의 변경 및 갱신에 대하여 클래스를 사용하여 논하였으나, 컴포넌트 사이의 의존 정보를 다루지 않고 있다.

2.2 컴포넌트 인터페이스

컴포넌트 인터페이스는 오퍼레이션들의 집합이다[6]. 또한 인터페이스는 컴포넌트와 분리된다. 각각의 오퍼레이션은 컴포넌트 객체가 사용자에게 제공하는 몇몇의 서비스나 기능들을 정의한다. 따라서 오퍼레이션은 사용자와 컴포넌트 객체 사이에서의 규약(Contract)을 나타낸다. 컴포넌트는 컴포넌트의 내부 구현으로부터 인터페이스의 명세를 완전히 분리하여야 한다. 즉, 컴포넌트 내부 구현이 변경되더라도, 기존의 컴포넌트 인터페이스 자체를 변경하지 않고도 컴포넌트의 서비스를 제공한다. 따라서 컴포넌트 인터페이스는 새로운 컴포넌트를 사용하거나 기존의 컴포넌트를 변경할 경우, 컴포넌트의 적응력 평가 시간을 감소시키고 테스트를 단순화 시킨다.



(그림 1) 컴포넌트 명세 다이어그램

2.3 컴포넌트 의존관계

컴포넌트 의존관계는 단순하게 표현된다. 즉, 두 사물간의 의미적인 관계로써 한쪽(독립) 사물의 변화가 다른(종속) 사물에 영향을 주는 관계를 의존관계라 하며, 컴포넌트 또는 클래스 사이의 의존 관계 표시를 나타내는 방법은 컴포넌트 명세 다이어그램에서 점선으로 표시한다. UML에서는 컴포넌트 명세와 인터페이스 타입 사이에 offers 관계가 형성되면, "<<offers>>"로 표시하고, 컴포넌트 명세와 인터페이스 타입 사이에 usage 관계가 형성되면, "<<usage>>"로 스테레오타입을 사용하여 구분한다[7].

이러한 컴포넌트 의존 관계는 컴포넌트 명세서에서 상세하게 명세되는데, 여기에는 컴포넌트 상호작용의 제약조건 및 컴포넌트 인터페이스 사이의 제약조건, 컴포넌트 내부 클래스도, 컴포넌트 인터페이스 명세서 등을 명세하여 개발자에게 제공된다. 그러나 컴포넌트 명세서가 제공되지 않을 때에는 컴포넌트 의존 관계 정보를 알지 못하는 상황에서 컴포넌트 합성 작업을 실시하게 되는 경우가 발생한다.

3. 컴포넌트 의존관계 명세

3.1 컴포넌트 명세

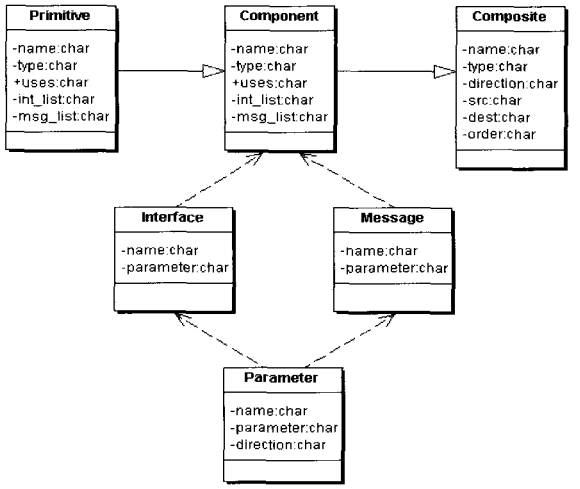
소프트웨어 시스템을 하나의 컴포넌트 조합으로 설계하고 구현하는 것이 규모가 큰 소프트웨어 개발을 더욱 적절하게 검색, 검증, 통합과정을 통하여 이룰 수 있게 되었다. 이러한 컴포넌트들의 재사용을 위해 필요한 것이 컴포넌트의 기능을 재사용 가능하게 명세하는 행위이다. 본 논문에서는 의존관계 명세를 재사용 가능하고 변경이 용이한 XML 형태로 정의하였다.

원시(Primitive) 컴포넌트는 패턴의 명세와 아키텍처의 명세에서 특정 컴포넌트에 의해 기능을 가지는 경우 필요한 기능을 지니는 컴포넌트 명세에 사용된다. 따라서 원시 컴포넌트는 합성 컴포넌트와는 달리 세부적인 컴포넌트 명세가 필요한 컴포넌트이다.

```
<component>
  <type> primitive </type>
  <name> 원시 컴포넌트 이름 </name>
        컴포넌트 명세 기술
</component>
```

(그림 2) 원시 컴포넌트 명세

합성(Composite) 컴포넌트는 원시 컴포넌트와는 달리 소프트웨어의 설계 패턴이나 아키텍처와 동일한 개념으로 정의되므로 다수의 컴포넌트와 커넥터의 조합으로 구성된다. 합성 컴포넌트 명세를 위해서는 일반적인 원시 컴포넌트와 컴포넌트 검색의 경우 원시 컴포넌트 명세, 그리고 커넥터 명세를 포함한다.



(그림 3) 컴포넌트 명세를 위한 클래스 다이어그램

컴포넌트 명세 언어에서 인터페이스는 CORBA 컴포넌트의 표준 인터페이스 정의언어인 IDL(Interface Definition Language) 명세 형태를 따른다. 인터페이스 명세는 해당 컴포넌트가 발생시킬 수 있는 메시지들의 목록을 기술한다. 메시지가 포함하는 파라미터 수와 타입 그리고 파라미터 방향성과 같은 정보를 나타내는 시그니처(Signature) 명세를 정의하였고, 인터페이스에 접근 가능한 컴포넌트 기능을 정의하여 기능 부분에 관한 내용을 정의하고, 이를 이용하여 전체적인 컴포넌트 인터페이스 명세를 정의하였다.

```
<signature>
  <return_type></return_type>
  <name></name>
  <param_list><num_of_param></num_of_param>
    <parameter>
      <type></type><name></name>
      <direct></direct>
    </parameter>
  <param_list>
</signature>
```

(그림 4) 인터페이스 시그니처 명세

```
<interface_list>
  <num_of_interface></num_of_interface>
  <interface>
    <name></name>
    <signature>
      <return_type></return_type><name></name>
      <param_list>
        <num_of_param></num_of_param>
        <parameter>
          <type></type><name></name>
          <direct></direct>
        </parameter>
      <param_list>
    </signature>
    <behavior>
      <name></name><modifies></modifies>
      <ensure></ensure>
    </behavior>
  </interface>
</interface_list>
```

(그림 5) 인터페이스 명세

컴포넌트는 메시지 정보를 교환하기 때문에 메시지 명세를 정의하는 것은 필수이다. 따라서 메시지 명세에는 커넥터에 의한 컴포넌트 사이의 통신에서 해당 컴포넌트가 다른 컴포넌트로부터 수신 가능한 메시지 목록에 대한 명세를 포함하며, 컴포넌트 명세에서 메시지 명세에 삽입하여 시그니처 정보를 이용하여 정의한다.

```
<message_list>
  <num_of_message></num_of_message>
  <message>
    <name></name>
    <signature>
      <name></name>
      <param_list>
        <num_of_param></num_of_param>
        <parameter>
          <type></type><name></name>
          <direct></direct>
        </parameter>
      <param_list>
    </signature>
  </message>
</message_list>
```

(그림 6) 메시지 명세

3.2 컴포넌트 의존명세

합성된 컴포넌트 사이의 의존관계 정보 명세를 정의하기 위하여 의존정보의 방향(Direction) 및 분류(Type) 정보로 구분하여 정의하고, 이를 메시지 명세에 삽입하여 정의한다.

```
<message_list>
    <signature>
      <depend>
        <direct></direct><type></type>
      </depend>
    </signature>
</message_list>
```

(그림 7) 의존정보 명세

3.3 컴포넌트 명세 구조

컴포넌트 명세 언어에서 정의된 XML은 사용자 태그가 유효하도록 하고, 각각의 태그에 대한 데이터 타입을 정의함으로써 사용자가 작성한 명세를 파싱(Parsing)하도록 한다. 컴포넌트 명세 언어를 정의하기 전에 XML의 특성상 DTD에 대한 정의를 하였다.

```
<!ELEMENT component (type,name,int_list?,msg_list*,uses*)>
<!ELEMENT int_list (num_of_int*, interface*)>
<!ELEMENT msg_list (num_of_msg*, message*)>
<!ELEMENT interface (name?, signature?, behavior?)>
<!ELEMENT message (name?, signature?)>
<!ELEMENT signature (name*, return_type*, param_list*)>
<!ELEMENT implementation (uses?, establish?)>
<!ELEMENT parameter (type?, name?, direct?)>
<!ELEMENT behavior (name?, modifies?, ensures?)>
<!ELEMENT depend (type*, direct*)>
<!ELEMENT name (#CDATA)>
<!ELEMENT type (#CDATA)>
<!ELEMENT uses (#CDATA)>
<!ELEMENT num_of_int (#PCDATA)>
<!ELEMENT num_of_msg (#PCDATA)>
<!ELEMENT return_type (#PCDATA)>
<!ELEMENT num_of_param (#PCDATA)>
<!ELEMENT direct (#PCDATA)>
<!ELEMENT modifies (#CDATA)>
<!ELEMENT ensures (#CDATA)>
```

(그림 8) 컴포넌트 명세 DTD 구조

컴포넌트 명세의 DTD를 바탕으로 하여 이전 항에서 정의한 명세 언어의 구조를 다음과 같이 종합하여 정의한다.

```
<component>
  <type></type><name></name>
  <interface_list>
    <num_of_interface></num_of_interface>
    <interface>
      <name></name>
      <signature><name></name>
        <return_type></return_type>
        <param_list>
          <num_of_param></num_of_param>
          <parameter>
            <type></type><name></name>
            <direct></direct>
          </parameter>
        </param_list>
      </signature>
      <behavior>
        <name></name><modifies></modifies>
        <ensure></ensure>
      </behavior>
    </interface>
  </interface_list>
  <message_list>
    <num_of_message></num_of_message>
    <message><name></name>
      <signature><name></name>
        <param_list>
          <num_of_param></num_of_param>
          <parameter>
            <type></type><name></name>
            <direct></direct>
          </parameter>
          <depend>
            <direct></direct><type></type>
          </depend>
        </param_list>
      </signature>
    </message>
  </message_list>
</component>
```

(그림 9) 컴포넌트 명세 XML 구조

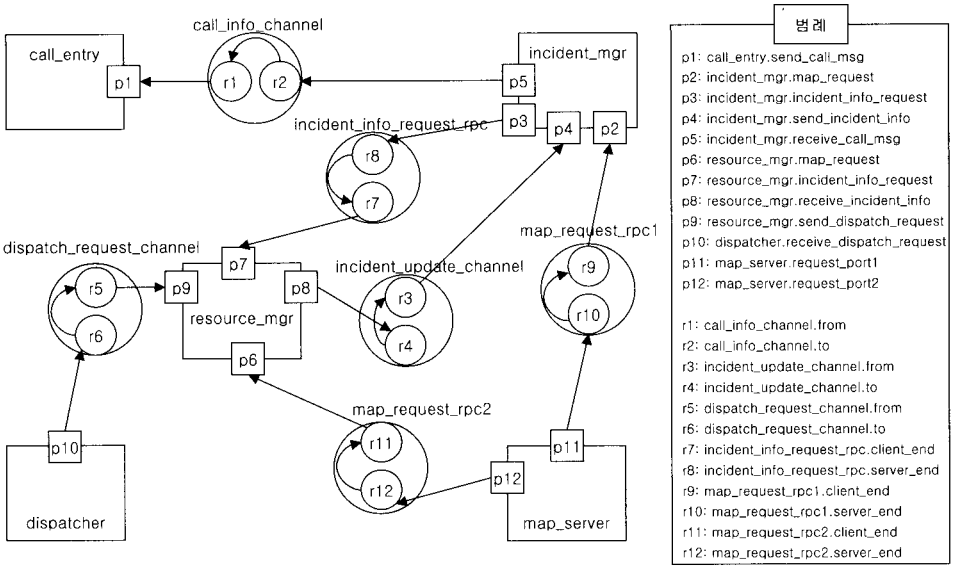
이상과 같이 정의한 컴포넌트 명세는 <component> 태그를 최상위 태그로 정의하였고, 각각의 하위 태그들을 계층적으로 표현함으로써 개발자가 그 의미를 파악하고 데이터를 분석하는데 어려움이 없도록 하였다. 또한 XML의 특징인 구조적인 데이터 표현을 컴포넌트 명세에 활용함으로써 개발자가 사용자 인터페이스의 질의에 기술하고자 하는 데이터만을 정의하면 XML 문서는 기본구조에 따라 자동으로 생성되도록 하였다.

4. 컴포넌트 의존성 분석

아키텍처 기반의 소프트웨어 개발을 지원하는 방법 가운데 중요한 기법이 의존성 분석 기법을 사용하는 것이다. 전통적인 프로그램 의존성은 프로그램 안에서 프로그램 문장 간의 의존 소유관계를 다루는 것이었는데, 이는 통제와 데이터 흐름에 의하여 결정되었다. 프로그램의 의존성을 결정짓는 타스크는 프로그램 의존분석을 통하여 이루어졌다. 이러한 전통적인 의존성 분석은 프로그램 의존성 그래프를 사용하여 정형적 방법으로 실시되었다. 시스템의 아키텍처적인 명세는 개발 단계에서 유용하게 사용될 것이고, 향후 시스템의 확장에 따른 노력을 덜 수 있게 정보를 제공할 것이다. 정확한 매핑이 아키텍처 명세와 구현 사이에서 유지되면, 아키텍처 명세의 수준 높은 특성은 구현의 다루기 쉬운 분석들을 실행하는데 기반한 신뢰성 있는 정보를 제공할 수 있다. 아키텍처 명세를 근거로 하는 의존성 분석은 아키텍처를 사용했던 언어가 세련되게 작성되도록 직접적인 영향을 미치는 경우도 있었다. 의존성 분석은 간단하게 소프트웨어 아키텍처의 "box-arrow"를 이용한 다이어그램으로 표현할 수 있다.

다음의 예는 컴포넌트 의존성 분석을 위하여 사용한 간단한 예로, 응급 구급 시스템의 구조를 보이고 있는데 재난신고(call_entry), 사고관리(incident_mgr), 자원관리(resource_mgr), 파견(dispatcher), 지도관리(map_server)의 5개 컴포넌트와 이들을 연결하는 채널 및 원격 호출 등을 갖는 6개의 커넥터(Connecter)로 구성된다.

다음 그림에서 call_entry와 incident_mgr 컴포넌트는 call_info_channel 커넥터로 연결되어있고, incident_mgr과



(그림 10) 응급 구급 시스템의 아키텍처

map_server 컴포넌트 사이는 map_request_rpc1 커넥터로 연결되는 등 모두 6개의 커넥터가 5개의 컴포넌트를 연결하고 있는 상태를 볼 수 있다. 아키텍처 수준에서의 컴포넌트와 커넥터 사이의 상호작용은 인터페이스를 통하여 이루어지는데, 컴포넌트의 포트(Port)와 커넥터의 롤(Role)에서 정의된다. 아키텍처 수준에서의 컴포넌트와 커넥터 사이의 의존성 분석을 두 가지로 구분한다.

4.1 컴포넌트-커넥터 의존성

이 의존성은 아키텍처 명세에서 컴포넌트의 포트와 커넥터의 롤 사이에 의존관계를 표현하는데 사용된다. (그림 10)에서 컴포넌트 resource_mgr의 receive_incident_info 포트(p8)와 커넥터 incident_update_channel의 to 롤(r4) 사이에 컴포넌트-커넥터 의존성이 존재한다. 이 경우에 메시지가 to 롤(r4)로 부터 receive_incident_info 포트(p8)로 전달된다.

4.2 커넥터-컴포넌트 의존성

이 의존성은 커넥터의 롤과 컴포넌트의 포트 사이에서 발생하는 의존관계를 나타낸다. (그림 10)에서 커넥터 call_info_channel의 from 롤(r1)과 컴포넌트 call_entry의 send_call_msg 포트(p1) 사이에 커넥터-컴포넌트 의존성이 존재한다. 이 경우 메시지가 send_call_msg 포트(p1)로부터 from롤(r1)로 전달된다.

아키텍처 수준에서 컴포넌트 사이의 의존관계를 이해하기 쉽게 하기 위하여 의존 테이블을 작성한다. 이 테이블은 합성된 컴포넌트에서 모든 의존관련 사항을 나타낸다. 즉 의존 테이블은 각각의 예제 컴포넌트가 서로 관계하여 의존하고 있는 상황을 보여준다.

의존 테이블에서 컴포넌트의 포트 p5 → p1, p3 → p7, p8 → p4, p12 → p6, p10 → p9, p11 → p2의 의존관계를 보여주고 있다.

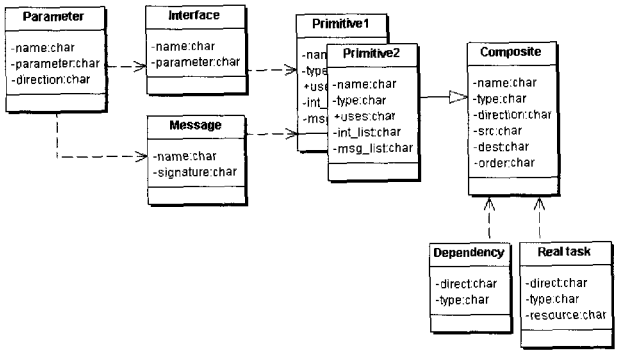
	call_entry	incident_mgr				resource_mgr				dispatcher	map_server	
	out	in		out		in		out		in	in	
	p1	p3	p5	p2	p4	p8	p6	p7	p9	p10	p11	p12
call_entry												
out												
p1												
incident_mgr												
in												
p3								○				
p5	○											
out												
p2												
p4												
resource_mgr												
in												
p8					○							
out												
p6												
p7												
p9												
dispatcher												
in												
p10									○			
map_server												
in												
p11				○								
p12							○					

(그림 11) 응급 구급 시스템의 의존 테이블

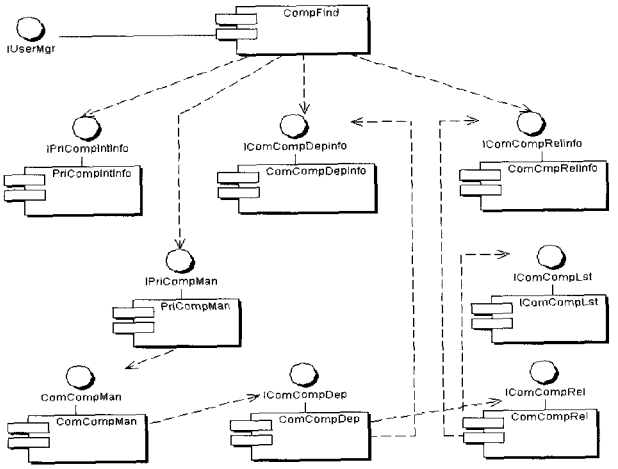
5. 컴포넌트 의존 관리자

컴포넌트 의존 관리자는 원시 컴포넌트의 인터페이스 정

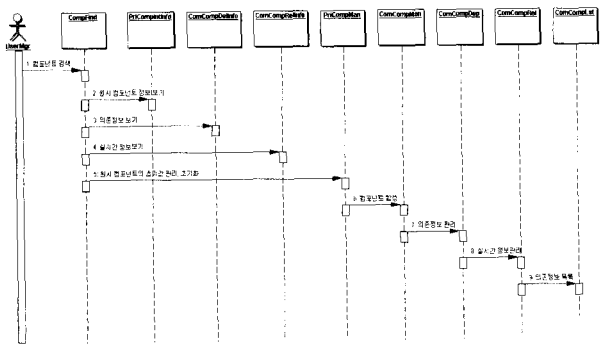
보와 메시지 정보를 확인할 수 있는데, 인터페이스 정보와 메시지 정보의 확인 후 원시 컴포넌트를 합성하도록 지원한다. 합성 컴포넌트는 컴포넌트 사이의 의존정보 포함하는 새로운 컴포넌트가 생성되는 것이다. 이 컴포넌트에는 원시 컴포넌트에는 가지지 않는 정보를 가지고 있기 때문에 명세 또한 복잡하게 구성된다. 합성 컴포넌트의 의존정보 및 실시간 지원 정보는 컴포넌트 의존 관리자에서 확인할 수 있다. 다음은 컴포넌트 의존 관리자의 클래스 다이어그램, 컴포넌트 다이어그램 및 순차 다이어그램을 나타내고 있다.



(그림 12) 의존 관리자 클래스 다이어그램

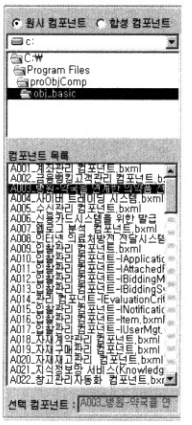


(그림 13) 의존 관리자 컴포넌트 다이어그램

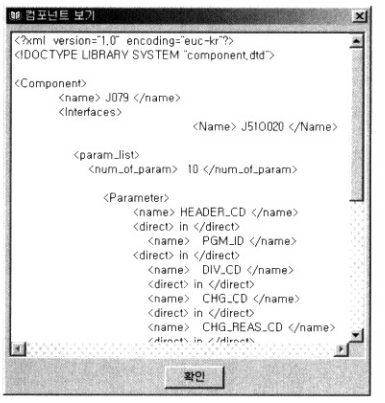


(그림 14) 의존 관리자 순차 다이어그램

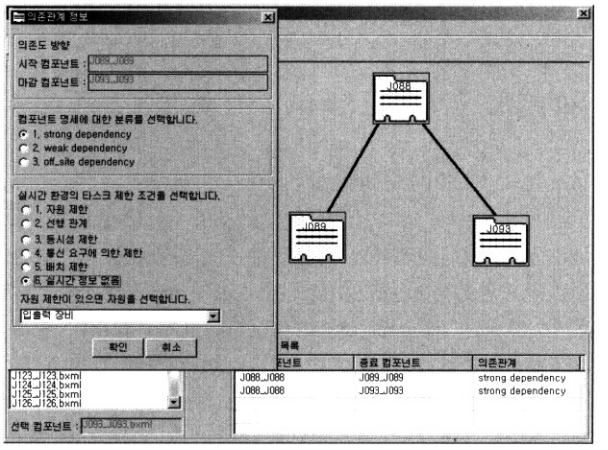
의존 관리자는 원시 컴포넌트의 검색과 검색된 컴포넌트의 인터페이스 명세 확인 등과 같은 컴포넌트 정보를 관리하고, 컴포넌트가 합성되는 경우 의존 관련 정보를 관리한다. 또한 합성된 컴포넌트의 명세를 확인하는 역할 역시 중요한 기능으로 작용한다.



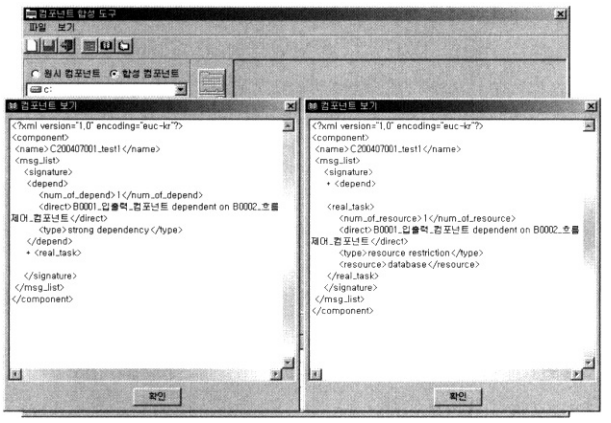
(그림 15) 원시 컴포넌트의 검색



(그림 16) 원시 컴포넌트의 인터페이스 명세



(그림 17) 컴포넌트 합성과정 및 의존정보 선택



(그림 18) 컴포넌트 합성과정 및 의존정보 확인

합성된 컴포넌트의 의존관계 명세 정보는 컴포넌트의 합성 후 의존 관리자를 통하여 정의되고 생성된다.

① 원시 컴포넌트 판단
② 컴포넌트 의존 관계의 분류(offer, usage) 정의
③ 컴포넌트 의존 분류 정의
④ 컴포넌트 포트 정보 확인
⑤ 컴포넌트 의존관계 정보 생성

또한 컴포넌트의 합성은 커넥터 정보의 일치를 통하여 이루어지고, 입출력 포트의 완전한 일치를 통하여 이루어지는데, 본 연구에서는 포트의 일치여부를 판단한 후 컴포넌트의 합성 작업을 하도록 구현하였다. 향후 연구에서 포트의 불일치 정보라도 이를 수정 후 합성이 가능하도록 추가 연구가 필요하다.

6. 평 가

합성된 컴포넌트의 평가를 위하여 컴포넌트의 복잡도 측정 메트릭을 이용하여 의존정보를 부여하기 이전과 이후의 복잡도를 측정하였고, 컴포넌트의 재사용성을 평가하기 위하여 컴포넌트 재사용성 측정 메트릭을 이용하여 측정하였다.

6.1 복잡도

컴포넌트는 재사용성과 기능 독립성을 목적으로 하는 소프트웨어 단위이므로 컴포넌트의 품질을 측정하기 위한 내부 속성 중에는 복잡도(Complexity)의 측정이 중요하다. 소프트웨어 복잡도는 컴퓨터 프로그램을 개발하고 유지하는 비용에 영향을 주는 소프트웨어 특성의 규명, 분류, 측정과 관련된 소프트웨어 공학의 한 영역이다[8]. 본 연구에서는 [9]에서 제안한 복잡도 측정 메트릭스를 이용한다. 이 메트릭은 설계 단위의 컴포넌트나 실행 단계의 컴포넌트에 대해 적용 가능한 컴포넌트 메트릭으로 이는 복잡도 측정을 위한 컴포넌트 일반 복잡도 메트릭으로서 클래스, 추상화클래스와 인터페이스의 합을 계산함으로써 컴포넌트의 복잡도 그 자체를 측정하며, 클래스들과 메소드들의 복잡도를 측정하는 기준이다[9].

$$CPC(C) = CmpC + \sum_{i=1}^{m} CCi + \sum_{j=1}^{n} MCj$$

여기에서 *CPC(C)*는 컴포넌트 일반 복잡도(Component Plain Complexity)이고, *CmpC*는 컴포넌트를 구성하는 클래스 수, 그리고 클래스 메소드 수에 의해 계산되며, 자신 컴포넌트에 속한 클래스는 컴포넌트를 계산할 경우 마찬가지로 개발되어야 할 부분이므로, 컴포넌트 내부에 속한 클래스나 그 클래스의 메소드에는 가중치를 주어 계산을 하여 컴포넌트의 외형적인 복잡도를 측정한다. 즉, *CmpC*의 계산은 다음과 같다.

$$CmpC = \sum_{i=1}^{m}(Count(Ci) \times W(Ci)) + \sum_{j=1}^{n} Count(Ij) + \sum_{k=1}^{\delta}(Count(Mk) \times W(Mk))$$

여기에서 *Count(C)*는 컴포넌트에 포함된 클래스의 수이고, *W(C)*는 각 클래스의 가중치이며, *I(j)*는 컴포넌트에서 사용되거나 제공되는 인터페이스이며, *Count(M)*는 컴포넌트에 포함된 클래스들의 메소드들의 수, 그리고 *W(M)*은 각 메소드의 가중치이다. $\sum_{i=1}^{m} CCi$*(CC)*는 컴포넌트 내부 각 클래스의 복잡도로 클래스의 속성을 단순(Single) 타입 속성인지, 다른 클래스를 타입으로 사용하는 복합(Complex) 타입 속성인지 구별하여 가중치를 주어, 컴포넌트 내의 클래스들의 복잡도를 구한다. $\sum_{j=1}^{n} MCj$*(MC)*는 컴포넌트 내부의 클래스들의 메소드에 대해 매개변수의 수가 얼마나 되며, 매개변수의 속성이 단순 타입인지, 다른 클래스를 타입으로 사용하는 복합 타입인지로 구별하여, 복합 타입의 매개변수에 마찬가지로 가중치를 주어 컴포넌트 내부 클래스의 메소드들의 복잡도를 구한다.

$$CC = \sum_{i=1}^{m} Count(SAi) + \sum_{j=1}^{n}(Count(CAj) \times W(CAj))$$

여기에서, *Count(SA)*는 개별 속성의 수이고, *Count(CA)*는 복합 속성의 수이며, *W(CA)*는 각각의 복합 속성의 가중치이다.

$$MC = \sum_{i=1}^{m} Count(SPi) + \sum_{j=1}^{n}(Count(CPj) \times W(CPj))$$

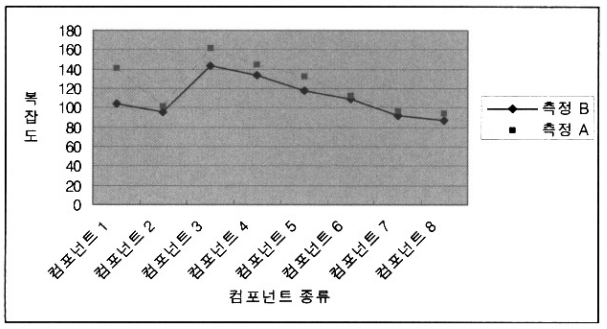
여기에서, *Count(SP)*는 단순 매개변수의 수이고, *Count(CP)*는 복합 매개변수의 수, 그리고 *W(CP)*는 각 각의 매개변수의 가중치이다.

이상과 같은 복잡도 측정 메트릭을 이용하여 의존정보를 부여하지 않았을 경우(A)와 의존정보를 부여(B)한 합성된 컴포넌트의 복잡도 측정을 실시하였다. 측정한 합성 컴포넌트는 총 8개로 각각 최종 사용자가 사용할 수 있는 컴포넌트이다. 컴포넌트 1의 B의 측정값은 CPC(C) = 32+55+17 = 104인데, 여기서, CmpC = 9+2+21 = 32이고, $\sum_{i=1}^{m} CCi$(CC) = 31+12*2 = 55이며, $\sum_{j=1}^{n} MCj$ = 7+5*2 = 17이다. 또한 컴포넌트 1의 A의 측정값은 141로 측정되었다.

컴포넌트 복잡도 측정 결과를 보면, 의존정보를 부여했을 때가 부여하지 않았을 때보다 각각 3에서 37까지 감소한 결과를 보이고 있다.

〈표 1〉 컴포넌트 복잡도 측정값

컴포넌트 구분	복잡도(A)	복잡도(B)
컴포넌트 1	141	104
컴포넌트 2	102	95
컴포넌트 3	162	143
컴포넌트 4	145	134
컴포넌트 5	132	117
컴포넌트 6	113	109
컴포넌트 7	97	92
컴포넌트 8	94	87

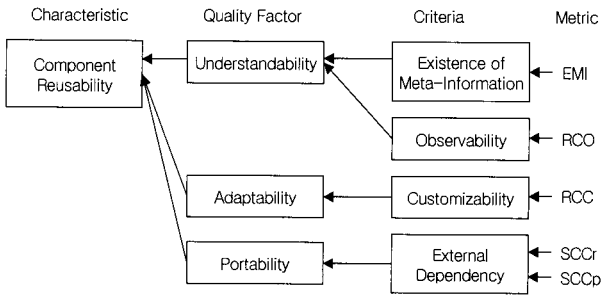


(그림 19) 컴포넌트 복잡도 비교

6.2 재사용성

소프트웨어의 품질을 측정하는 모델을 소프트웨어 품질 모델이라 하고, 대표적인 모델로는 ISO/IEC 9126, McCall의 품질모델, Boehm의 품질모델 및 Dromey의 품질모델이 있다. ISO/IEC 9126에서 제시한 범용적 품질모델을 기반으로 하여 컴포넌트 소프트웨어에 적용할 수 있는 특화된 품질 모델을 정의하기 위하여 품질요소를 기능성, 재사용성, 준수성으로 구분하여 정의하였다. 이 가운데 컴포넌트 재사용은 컴포넌트 품질특성 가운데 중요한 특성 중의 하나로 컴포넌트에 대한 재사용이 가능한지와, 직접 개발하는 요소는 줄이고 다른 컴포넌트들과 조립함으로써 얼마나 많은 개발비용을 줄일 수 있는지를 판단하는 메트릭이다[10].

ISO/IEC 9126 품질 요소 가운데 컴포넌트 소프트웨어의 재사용성은 McCall의 Factor-Criteria-Metrics(FCM) 모델을 적용하여 사용한다[10,11]. FCM 모델은 소프트웨어 평가를 위한 ISO 9126 모델에 적용된 접근방법으로 블랙박스 컴포넌트의 재사용성 측정 메트릭 모델로, Understandability, Adaptability, Portability의 3개 Quality factor로 구성된다[11]. Understandability는 컴포넌트가 제공하는 기능적인 요소에 대한 이해, 즉 컴포넌트의 재사용 시 기능적인 요구사항에 대해 컴포넌트가 이를 만족시킬 수 있는가 없는가를 결정할 수 있게 어느 정도의 정보를 제공하는 가에 대한 메트릭이다. Adaptability는 원래 의도한 대로 개발된 컴포넌트로부터 원래의 요구사항과는 다른 새로운 시스템에서의 특정한 기능적인 요구사항에 적용시킬 수 있는가에 대한 메트릭이다. Portability는 컴포넌트에 대한 포팅 능력에 대한



(그림 20) 컴포넌트 재사용성 메트릭 요소

메트릭, 즉 어떤 환경에서 새로운 환경으로의 변환이 어느 정도 쉽게 가능한지에 대한 메트릭이다[11].

6.2.1 Existence of meta-information

이 메트릭은 목표 컴포넌트가 메타 정보를 제공하는지에 대한 평가항목으로 컴포넌트의 재사용시 기능적인 요구사항에 대하여 컴포넌트가 이를 만족시킬 수 있는지의 정보를 제공하는가에 대한 메트릭이다.

$$EMI = \begin{cases} 1 & (Classexists) \\ 0 & (Otherwise) \end{cases}$$

이 평가 항목은 해당 클래스가 존재하면 측정값이 1로 이는 컴포넌트를 사용할 때 쉽게 이해할 수 있고, 존재하지 않으면 0으로 이해성이 없다는 것이다. 본 연구에서 평가하는 컴포넌트는 모두 메타 정보를 갖는 클래스를 포함하므로 EMI 측정 메트릭은 사용하지 않는다.

6.2.2 Observability

이 메트릭은 컴포넌트가 가진 기능적인 행위요소 및 입출력 파라미터에 대해서 얼마나 쉽게 볼 수 있는지에 대한 평가항목이다. 컴포넌트의 행위는 컴포넌트의 프로퍼티에 대응하는 읽기 메소드를 사용하기 때문에 컴포넌트가 제공하는 읽기 메소드의 수가 중요한 요소가 된다. 이 평가 항목은 Rate of Component Observability로 표기된다.

$$RCO = \begin{cases} \frac{P_r}{A} & (A > 0) \\ 0 & (Otherwise) \end{cases}$$

여기에서 P_r은 읽기 가능한 프로퍼티의 수이고, A는 전체 컴포넌트 클래스 필드, 즉 프로퍼티의 수이다.

6.2.3 Customizability

이 메트릭은 컴포넌트 내부의 기능적인 특성에 대한 최적화 및 형상을 지원하는지에 대한 평가 항목으로, 프로퍼티는 최적화 및 형상이 사용된다. 따라서 컴포넌트는 프로퍼티 값이 인터페이스의 타입에 의해 변경된다. 이 평가 항목은 Rate of Component Customizability로 표기된다.

$$RCE = \begin{cases} \frac{P_w}{A} & (A > 0) \\ 0 & (Otherwise) \end{cases}$$

여기에서 P_w는 쓰기 가능한 포로퍼티의 수이고, A는 전체 컴포넌트 클래스 필드, 즉 프로퍼티의 수이다.

6.2.4 External dependency

외부 의존은 비즈니스 메소드와 같이 컴포넌트 외부로부터 호출되어지는 메소드에서의 리턴값이나 파라미터를 통한 외부 의존성에 대한 평가항목을 나타낸다. 이 메트릭은 Self-Completeness of Component's Value와 Self-Completeness of Component's Parameter로 표기된다.

$$SCV = \begin{cases} \frac{B_v}{B} & (B > 0) \\ 0 & (Otherwise) \end{cases}$$

여기에서 B_v는 리턴값이 없는 비즈니스 도메인의 수를 나타내고, B는 전체 컴포넌트 비즈니스 메소드의 수를 나타낸다.

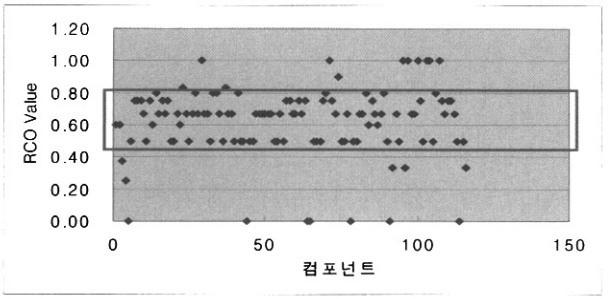
$$SCP = \begin{cases} \frac{B_p}{B} & (B > 0) \\ 0 & (Otherwise) \end{cases}$$

여기에서 B_p는 파라메터가 없는 비즈니스 도메인의 수를 나타내고, B는 전체 컴포넌트 비즈니스 메소드의 수를 나타낸다. 컴포넌트의 재사용 가능한 상태를 판정하기 위하여, 합성된 116개의 JavaBeans 컴포넌트를 측정하여 다음과 같은 결과를 얻었다.

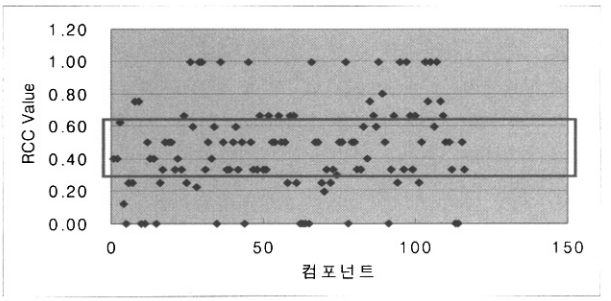
각 측정값의 분포 및 신뢰도를 나타내기 위하여 측정 메트릭별로 97%의 신뢰구간을 설정하여 RCO는 (0.43, 0.81), RCE는 (0.29, 0.63), SCV는 (0.52, 0.82), SCP는 (0.26, 0.72)로 구하였는데, 측정한 100개의 데이터 그룹에서 각 그룹별 신뢰구간을 구했을 때, 97%는 신뢰구간 범위에 포함될 확률을 가지는 범위의 값이라고 해석할 수 있다. 다음은 측정값의 분포를 히스토그램으로 나타내고 있다.

〈표 2〉 컴포넌트 재사용 메트릭 측정값

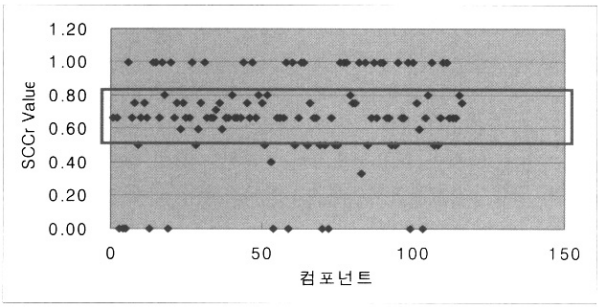
컴포넌트	RCO			RCE			SCV			SCP		
	값	Pr	A	값	Pw	A	값	Bv	B	값	Bp	B
TA001	0.60	3	5	0.40	2	5	0.67	2	3	0.67	2	3
TA002	0.60	3	5	0.40	2	5	0.67	2	3	0.67	2	3
TA003	0.38	3	8	0.63	5	8	0.00	0	0	0.00	0	0
.....												
TJ197	0.50	1	2	0.50	1	2	0.80	4	5	0.40	2	5
TJ198	0.33	1	3	0.33	1	3	0.75	3	4	0.50	2	4
해당 컴포넌트	109			102			105			93		
평균	0.62			0.46			0.67			0.49		



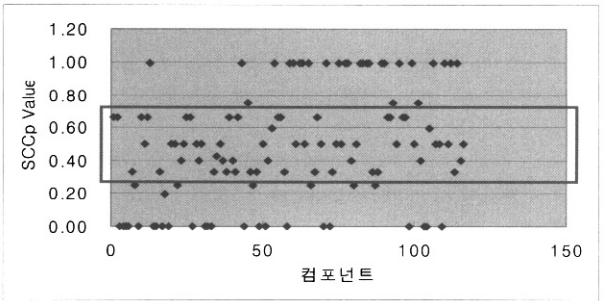
(그림 21) RCO에 대한 히스토그램



(그림 22) RCE에 대한 히스토그램



(그림 23) SCV에 대한 히스토그램



(그림 24) SCP에 대한 히스토그램

측정한 값의 분포를 보면 RCO는 1과 0을 제외한 값의 92%가 신뢰구간에 포함되어 있고, RCE는 89%, SCV는 93%, SCP는 95%가 신뢰구간에 포함되어 있는 것으로 측정되었다. 따라서 전체 측정값이 비교적 재사용성의 확률이 높다는 것을 보여주고 있다.

7. 결론 및 향후 연구방향

본 논문에서는 컴포넌트의 의존관계 정보를 제공할 수 있도록 컴포넌트의 명세를 재사용 가능하고 편집 가능한 XML로 정의하였고, 의존관계 정보를 포함하는 포괄적인 컴포넌트 명세를 XML 형태로 정의하였다. 컴포넌트는 상호 의존성을 가지고 있기 때문에 이들 관계를 파악하는 것은 중요한 일이므로 컴포넌트의 의존성을 테이블로 정의하여 관계를 파악하도록 의존 테이블을 작성하였다. 컴포넌트를 합성하는 과정에서 컴포넌트의 의존관계 정보를 설정하고 정보를 확인할 수 있도록 의존 관리자를 설계하여 구축하였다. 의존 관리자에 의하여 생성된 컴포넌트의 재사용성을 측정한 결과 신뢰구간 안에 89%에서 95%까지 분포되어 있어 재사용성이 높게 나타났다. 향후 연구방향으로는 재사용 가능한 컴포넌트의 합성이 뒤따르는 합성도구의 완전한 시스템화를 위한 작업이 이어져야 할 것이다.

참 고 문 헌

[1] Ivica Crnkovic, Magnus Larsson, "Component Configuration management", In ECOOP Conference, Workshop on Component Oriented Programming Nice, 2000.

[2] Hong Mei, Zhang Lu, Yang Fuqing, "A Software Configuration Management Model for Supporting Component-Based Software Development", ACM SIGSOFT Software Engineering Notes (USA), 26(2), 2001.

[3] 차정은, 김행곤, "CBD 프로세스 지원을 위한 컴포넌트 저장소의 구축", 정보과학회논문지 : 소프트웨어 및 응용, 제29권 제7호, pp.476-486, 2002.8.

[4] Magnus Larsson, Ivica Crnkovic, "Configuration Management for Component-based Systems", In Software Configuration Management(SCM-10), 23th ICSE Toronto, Canada, May, 2001.

[5] Xuejun Chen, "Dependence Management for Dynamic Reconfiguration of Component-Based Distributed Systems", Proceedings of the 17 th IEEE International Conference on Automated Software Engineering(ASE'02), September, 23-27, 2002.

[6] Desmond Francis D'Souza, Alan Cameron Wills, "Objects, Components, and Frameworks with UML", Addison-Wesley, 1998.

[7] Grady Booch, James Rumbaugh, Ivar Jacobson, "The Unified Modeling Language User Guide", Addison-Wesley, 1999.

[8] Roger S. Pressman, Software Engineering : A Practitioner's Approach, 5th edition, McGraw-Hill, 2002.

[9] E.S. Cho, M.S. Kim, M.S. Kim, "Component Metrics to Measure Component Quality", Eighth Asia-Pacific Software Engineering Conference (APSEC'01), pp.419-426, Dec., 04-07, 2001.

[10] H. Washizaki, Y.Yamamoto and Y. Fukazawa, "Software Component Metrics and It's Experimental Evaluation", Proc. of International Symposium on Empirical Software Engineering (ISESE2002), Vol.II, 2002.

[11] H. Washizaki, Y.Yamamoto and Y. Fukazawa, "A Metrics Suite for Measuring Reusability of Software Components", 9th IEEE International Symposium on Software Metrics, 2003.

[12] P. Feiler, "Configuration Management Models in Commercial Environments", Tech. Report. CMU/SEI-91-TR-7, Software Engineering Institute, Carnegie-Mellon University, Mar., 1991.

[13] J. M. Sagawa, "Repository Manager technology", IBM SYSTEMS JOURNAL, Vol.3, No.2, pp.209-227, 1990.

[14] Andre van der Hoek, Alexander L. Wolf, "Software Release Management for Component-Based Software", In Software-Practice and Experience 33, January, 2003.

[15] Miro Casanova et al, "Supporting Evolution in Component-Based Development using Component Libraries", Seventh European Conference on Software Maintenance and Reengineering (CSMR'03), March, 26-28, 2003.

[16] Fabio Kon, Roy Campbell, "Dependence Management in Component-Based Distributed Systems", IEEE Concurrency, 8(1), pp.26-36, January~March, 2000.

최 상 균

e-mail : skchoi@kimpo.ac.kr
1986년 한남대학교 전자계산공학과 (공학사)
1993년 서강대학교 정보처리학과 (이학석사)
2000년 경희대학교 전자계산공학과 박사과정 수료
1986년~1997년 한국생산기술연구원 선임연구원
1998년~현재 김포대학 컴퓨터계열 교수
관심분야 : CBSE, 소프트웨어 재공학, 소프트웨어 품질

송 영 재

e-mail : yjsong@khu.ac.kr
1969년 인하대학교 전자공학과(공학사)
1976년 일본 Keio 대학교 전산학과 (공학석사)
1980년 명지대학교 전산학과(공학박사)
1982년~1983년 미국 Maryland대학교 객원교수
1986년~1988년 대한전자공학회 전자계산연구회전문위원장
1984년~1989년 전국 전산소장 협의회 부회장
1990년~1991년 일본 Keio 대학교 객원교수
1984년~1989년 경희대학교 전자계산소장
1993년~1995년 경희대학교 교무처장
1996년~1998년 경희대학교 공과대학장
1999년~2000년 경희대학교 기획조정실장
2001년~2002년 경희대학교 산업정보대학원장
1976년~현재 경희대학교 전자정보학부 교수
관심분야 : 소프트웨어공학, CBSE, CASE 도구, S/W 재사용