특집

CBD 관리 기법

한정수[1] 고응남[1] 이정배[2]

목 차

1. 서 론

최근 소프트웨어 개발과정이 in-house 형태의 개발에서 표준화된 컴포넌트(standard components), 아웃소싱(outsourcing), 상용 컴포넌트(COTS: commercials off the shelf components) 등의 새로운 패러다임으로 진행되고 있다. 즉, 개발된 최종 시스템은 하나의 고정된 시스템이 아니라 컴포넌트 기반으로 개발되어 다른 시스템과 통합될 수 있고, 자체적으로 갱신(upgrade)될 수 있는 개방형(open) 또는 유동성(flexible) 있는 시스템으로 발전되고 있다. 이와 같은 시스템에서는 시스템이 실시간으로 업그레이드되기 때문에 시스템 형상(configuration)에 영향을 준다. 따라서 이러한 패러다임은 개발의 효율성을 증가시키지만 시스템 형상의 일관성 또는 신뢰성을 감소시키는 위험이 따른다[1,4].

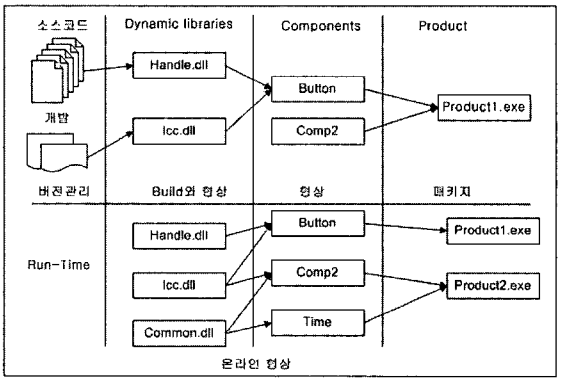
컴포넌트 형상관리(CCM : component configuration management)는 시스템 내부의 구성원들 사이의 일치성을 제어함으로써 신뢰도를 증가시키는 역할을 한다. 컴포넌트 기반으로 개발된 패키지에 새로운 컴포넌트가 추가 또는 대체될 때 두 가지 문제점을 해결해야 한다. 첫째 기존의 컴포넌트가 다른 패키지와 연결되어 있을 경우 이를 대체 시킬 때 발생하는 문제점이다. 즉, 새로운 버전의 컴포넌트가 대체될 때 변경(changes)과 컴포넌트들 사이의 관계가 불확실하다는 것이다. 둘째 실시간 환경에서의 시스템에 대한 동적 행위는 더 심각한 문제점이다. 즉, 실시간으로 컴포넌트가 대체되면 하나의 패키지는 동작을 하지만 이전의 컴포넌트와 연결된 패키지는 동작하지 않을 수 있다는 것이다. 이러한 문제점의 해결방안이 바로 컴포넌트 형상관리(CCM)다. 따라서 본 연구에서는 컴포넌트 관리에 대하여 분석해보고, 컴포넌트 형상에 관련된 문제점들을 지적하며, 이러한 문제점을 해결할 수 있는 컴포넌트 형상관리 방향에 관하여 기술하였다.

2. CBD LifeCycle

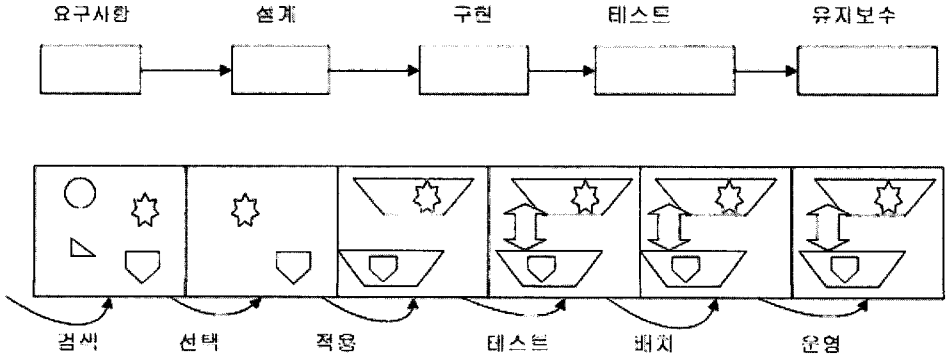
소프트웨어 공학에서 많은 방법론들이 소프트웨어 개발 생명주기를 유사하게 제시하고 있고 같은

1) 천안대학교 정보통신학부 교수
2) 선문대학교 교수

방식으로 사용되고 있다. 그러나 CBSE는 컴포넌트 개발과 컴포넌트를 이용한 시스템 개발 모두를 포함하고 있다. 기존의 방법들과의 차이점은 컴포넌트의 재사용성이다. 컴포넌트가 많은 애플리케이션에서 만들어지고 재사용되기 위해서는 명세화, 이해하기 쉽고, 범용성, 적응성, 재배치 등이 잘 이루어져야만 가능하다. 컴포넌트 개발은 설계에서 두 가지 단계를 고려해야 한다. 첫째, 컴포넌트 기능과 컴포넌트 사이의 관계에 대한 시스템 아키텍처 명세화다. 이는 시스템의 논리적 관점을 인식시켜준다. 둘째, 물리적으로 컴포넌트를 구성하는 시스템 아키텍처 명세화다. (그림 1)은 폭포수모델(waterfall model)과 비교한 CBD 생명주기다. CBD 생명주기는 컴포넌트를 검색하여 정확한 컴포넌트를 선택한 후 기존의 시스템에 적용하여 테스트한 후 재배치되는 과정을 말한다[2]. (그림 2)는 개발단계와 런타임에서의 형상관리를 보여주고 있다. 개발단계에서는 소스코드로부터 라이브러리를 만들고, 컴포넌트는 라이브러리(dll 파일)를 조립하여 생성한다. 그리고 컴포넌트 기반 프러덕트가 컴포넌트들의 집합으로 구현된다. 이 과정에서 첫 번째 개발단계에서의 소스코드 관리는 코드의 버전 정보를 이용하여 이루어진다. Build는 여러 가지 CM 도구(make and configuration tool))에 의해서 소스코드와 연결하여 진행되고, 끝으로 컴포넌트들의 조립으로 프러덕트가 생성된다. 이때는 소스코드를 제어하고, 이 코드로부터 전체 시스템을 만들기 때문에 시스템 형상관리를 제어할 수 있다. 그러나 하나의 컴포넌트가 추가, 또는 대체되면 이 제어는 컴포넌트의 기능을 부분적으로 알기 때문에 사라진다. 하지만 컴포넌트의 버전제어가 가능하다면 버전과 형상관리가 동시에 가능하기 때문에 컴포넌트 사이의 불확실한 관계를 해결할 수 있다.



(그림 2) 컴포넌트 기반 Product 생명주기에서 형상관리



(그림 1) 폭포수 모델과 비교한 컴포넌트 생명주기

3. 컴포넌트 형상 관리

3.1 컴포넌트 호환성

컴포넌트의 특성은 독립적이어야 하며 어떤 뚜렷한 기능을 갖으면서 시스템의 한 부분으로 대체될 수 있어야 한다. 런타임 컴포넌트는 실행 중에 인터페이스를 통하여 단위 프로그램으로 패키지에 동적으로 추가될 수 있어야만 독립적인 기능으로 시스템에 재배치가 가능한 컴포넌트의 기능을 다하는 것이다. 이처럼 컴포넌트의 중요성은 개발과 통합을 위한 기술이 표준화되어야 그 효력을 발휘한다. 현재 애플리케이션 개발의 컴포넌트 기술은 EJB(Enterprise Java Bean), COM+(Component Object Model), ActiveX, CORBA 등이 표준으로 사용되고 있다.
새로운 컴포넌트 버전이 시스템의 새로운 기능으로 추가되어 기능이 변경될 수 있다. 이때 고려해야 할 사항이 호환성(compatibility)이다. 호환성은 세 단계로 분류할 수 있다[3].

3.1.1 입출력 호환성(input and out compatibility)

컴포넌트는 특정 형식(format)으로 입력을 요구하여 정의된 형식으로 출력한다. 즉, 컴포넌트의 내부적 특성은 고려할 필요가 없다. 그 예로서 서로 다른 워드 프로세서가 같은 문서 형식을 출력하는 경우다.

3.1.2 인터페이스 호환성(interface compatibility)

개발기간이나 런타임 때 인터페이스는 같을 수 있지만 구현은 다른 경우다. 즉, 같은 인터페이스를 통하여 서로 다른 ActiveX object를 구현하는 경우다.

3.1.3 행위 호환성(behavior compatibility)

컴포넌트의 내부 특성은 보존되어야 한다. 이와 같은 보존성은 실시간 시스템에 적합하며 가장 중요한 요소다.

호환성은 컴포넌트가 대체될 수 있는지를 결정하는 기준이 되며, 이러한 결정은 런타임 환경에서 특히 중요한 요소가 된다. 즉, 전체 시스템의 인터럽트 위험을 피하기 위해서는 호환성을 위한 세 단계를 고려해야 한다.

3.2 컴포넌트 변경(component changes)

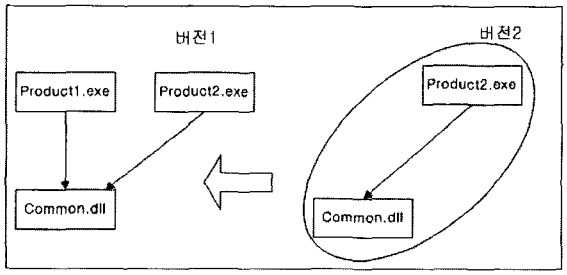
일반적으로 컴포넌트는 공유 라이브러리로 구성한다. 컴포넌트를 사용하는 프로그램은 직접 라이브러리를 참조하는 것이 아니라 컴포넌트 인터페이스를 참조한다. 즉, 라이브러리는 인터페이스의 구현이다. 여기서는 컴포넌트의 논리적, 물리적 단계의 변경 정보뿐만 아니라 관계의 정보까지 알아야 한다. 즉, 라이브러리와 인터페이스가 일치되어야 하기 때문에 컴포넌트 형상관리는 두 레벨(라이브러리, 인터페이스)에서 이루어져야 한다. 컴포넌트 관리를 위해서 고려해야 할 사항은 다음과 같고, 컴포넌트 변경을 라이브러리와 인터페이스로 나누어 기술한다.

- Manage library repository
- Manage library users
- Manage catalogues
- Assure quality components
- Manage component
- Make components available
- Manage the version of components

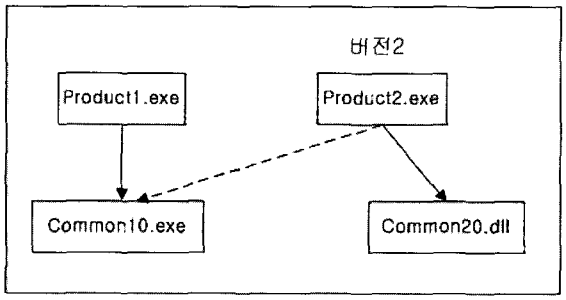
3.2.1 라이브러리(libraries)

일반적으로dll 파일들이 나오기 이전의 라이브러리는 정적으로 실행파일과 링크되어 있기 때문에 큰 문제점은 없다. 이것은 실행파일이 새로운 버전의 라이브러리로 갱신(update)될 수 없음을 의미한다. 즉, 실행파일을 제어 없는 사용으로부터 보호하는 것이다. 즉, 새로운 라이브러리를 사

용하기 위해서는 실행파일과 다시 링크해야하는 문제점이 발생한다. 이것은 라이브러리가 인터페이스 호환성을 갖는다는 의미에 위배된다. 또 다른 문제점은 갱신 이전의 라이브러리와 연결된 모든 실행파일들도 모두 재링크해야만 되는 것이다. 그러면 이의 해결 방안은 라이브러리는 인터페이스 호환성을 갖으면서 실행파일은 재링크를 하지 않고도 새로운 라이브러리를 공유하는 것이다. 이의 해결은 공유 라이브러리들을 dll(dynamic link libraries)로 설계하는 것이다. dll 파일은 실행파일이 필요할 때마다 호출하기 때문에 호환성이 가능하다. 그러나 dll 파일 역시 시스템의 일관성에 대한 새로운 문제점이 발견된다.



(그림 3) dll 변경에 의한 product1의 실행중단



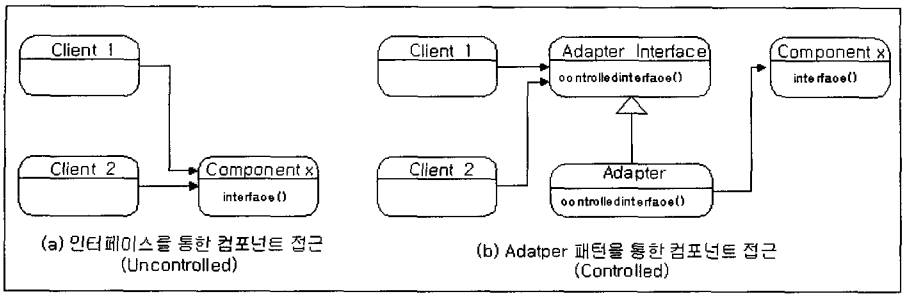
(그림 4) Common.dll의 공존

(그림 3)은 이 시스템에 Product2.exe가 업그레이드(upgrade)되면서 버전2의 Common.dll이 인터페이스 호환성을 갖고 있어서 버전1의 Common.dll로 대체될 경우 성공적이지만 이 시스템 내부에 연결되어 있는 Product1은 중단될 수 있음을 보여준다. 이의 해결책은 라이브러리의 다중 버전(multiple version)을 이용하는 것이다. 예를 들면 MFC40.dll, MFC42.dll과 같은 이름으로 사용하는 것이다. (그림 4)에서처럼 이름 충돌 문제를 해결할 수 있다. 그러나 이것 역시 많은 유사 버전들이 생성되면서 제어가 어려운 단점을 갖고 있다. 변경에 의해 영향을 받는 시스템 내부 구성원을 이해하기 위한 내용은 다음과 같다.

- 컴포넌트와 연결된 버전을 함께 인식
- 직접 또는 간접으로 연결된 의존성(dependencies)을 인식
- 의존범위에 대한 충분한 정보를 인식

3.2.2 인터페이스(interface)

인터페이스는 컴포넌트와 사용자 사이의 연결이다. 인터페이스의 역할은 컴포넌트 구현과 분리되는 것이 그 목적이다. 분리를 위하여 Adapter design pattern을 이용하면 해결할 수 있다[3]. (그림 5-a)에서 두 client가 하나의 컴포넌트 인터페이스에 의존한다. 컴포넌트와 인터페이스가 변경되면 두 client 모두가 영향을 받는다. Adapter design pattern을 사용하면 (그림 5-b)처럼 client와 컴포넌트사이의 정보 또는 컴포넌트의 인터페이스에 관련된 정보를 이용하여 변경이 가능하다. 즉, Adapter Interface를 통하여 컴포넌트에 접근할 수 있다. 여기서 Adapter는 컴포넌트 구현과 분리되며 하나의 객체(object)로 취급된다. 그러면 컴포넌트 의존은 Adapter와의 관계일 뿐이다. 따라서 컴포넌트가 새로운 버전이나 유사 컴포넌트로 변경될 때 Adapter만 갱신(update)시키면 시스템의 다른 컴포넌트에 영향을 최소화시키면서 목적 컴포넌트와 연결하여 업그레이드시킬 수 있다. 이것은 중복이나 통합의 노력이 필요하지만 컴포넌트 개발자보다는 시스템 운영자의 또는 사용자에 의하여 손쉽게 컴포넌트의 조립이 가능한 방법이다.



(그림 5) Adapter를 통한 컴포넌트 인터페이스 관리

3.3 컴포넌트 형상관리 방법

3.3.1 라이브러리 형상관리

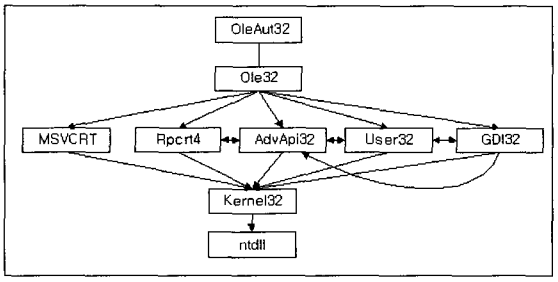
시스템이 새로운 프로그램으로 업그레이드될 때 수행중인 프로그램은 어떠한 경고 없이 영향을 받을 수 있다. 따라서 업그레이드되는 프로그램 또는 컴포넌트들에 대한 인터페이스가 필요하다. 공유 라이브러리가 어떤 라이브러리나 프로그램에 연결되어 있는지는 프로그램과 라이브러리 사이의 의존 목록을 이용하면 쉽게 알 수 있다. 라이브러리를 포함하는 새로운 프로그램을 설치할 때 다음의 단계로 진행하면 된다.

① 현 시스템의 형상에 대한 스냅샷(snapshot)을 얻는다

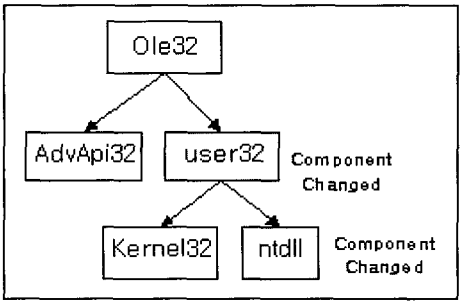
② 새로운 모듈을 설치한다.

③ 새로운 시스템의 형상에 대한 스냅샷(snapshot)을 얻는다.

스냅샷의 내용은 설치되는 모든 프로그램과 라이브러리에 관한 테이블이 된다. 각 속성에 대한 정보를 그래프로 표현한다면 (그림 6)과 같다. 각 속성들은 최근 날짜(date), 시간(time), 크기(size), 이름(name), 관계정보(rel_info) 등을 갖고 있다. 시스템의 스냅샷은 의존 그래프로 표현할 수 있다. (그림 6)은 COM 라이브러리의 OleAut32.dll에 대한 의존관계 그래프다. 스냅샷의 서로 다른 버전은 버전제어 하에 있으며 형상관리 품목으로 취급된다. 이러한 정보는 사용자에게 그 차이점들을 제공해 주며, 사용자는 새롭게 갱신된 라이브러리에 대하여 이해하게 된다. 라이브러리가 잘못 변경되면 그 결과를 알려준다. 이 정보들은 시스템 오류의 원인을 추적하기가 용이하다. (그림 7)은 변경된 버전의 그래프를 보여준다. 이 같은 정보는 어느 컴포넌트가 변경되고 그 변경에 의해서 어떤 컴포넌트가 영향을 받는지에 관한 직접적인 정보는 알 수 없지만 컴포넌트의 물리적 표현이 라이브러리이기 때문에 간접 정보는 유용하다.



(그림 6) OleAut32.dll에 대한 의존관계



(그림 7) 변경 후 의존관계

3.3.2 컴포넌트 형상관리

컴포넌트는 조립으로 구성된 복합체의 한 부분이다. 컴포넌트 관리는 소프트웨어 형상관리(SCM: Software Configuration Management)와 유사하다. SCM의 목적은 복합체를 구성하는 각 구성원(entities)들을 관리하는 것이다. 그러나 대부분의 SCM 기능은 개발과정에서 사용되지만 런타임 시에는 잘 활용되지 못하고 있다. SCM의 주요 원칙은 버전관리(version management), 형상관리(configuration management), 변경관리(change managem - ent)이다. 여기에 컴포넌트 관리를 위해서는 이 세 가지 원리와 의존관리(dependency management)가 필요하다[5].

3.3.2.1 버전관리(Version Management)

버전관리는 각 엔티티를 구별하여 서로 다른 버전의 엔티티를 인식하는 것이다. 이 원리는 런타임 때 컴포넌트에 적용할 수 있다. 시스템에 있는 모든 컴포넌트는 이름(name), 버전(version number), 기타 버전 속성(생성날짜 등) 등으로 인식되어야 한다. 여기서 컴포넌트 관리를 위해 컴포넌트 버전 ID가 필요하다. 그 이유는 첫째, 컴포넌트가 새로운 버전으로 갱신될 때 그 변경을 인식해야하기 때문이다. 둘째, 하나의 시스템에 통합된 여러 컴포넌트 버전들을 관리해야하기 때문이다. 서로 다른 컴포넌트들의 관리는 중간규모(middle-size) 또는 대규모(large scale) 시스템에서 매우 중요하다. 원래 컴포넌트라는 것은 시스템 전체의 요구사항을 포함시키도록 설계되는 것이 아니다. 즉, 많은 시간을 투자하여 전체적으로 완벽하게 만들어서 나오는 것이 아니라 현재 필요한 특성들을 갖는 컴포넌트를 개발하고 그 후에 업그레이드(upgrade)하는 것이 더 효율적이다. 새로운 특성이 컴포넌트에 추가될 때 이 버전이 이전의 버전과 호환되지 않을 경우도 발생한다. 따라서 두 버전 모두를 유지해야하며 동시에 시스템은 이러한 환경(동시에 여러 버전이 사용되는)을 모두 제공해야 한다.

3.3.2.2 형상과 빌드 관리(Configuration and Build Management)

형상과 빌드 관리는 특정 버전을 검색, 선택하여 통합시키는 과정이다. build는 의존관계 정보를 이용한다. 이 원칙을 런타임 시스템에 적용할 수 있다. 시스템 형상은 컴포넌트의 집합체이다. 따라서 컴포넌트가 추가되면 실제 컴포넌트는 이진 코드로 되어 있기 때문에 컴포넌트는 명세서(specification)를 갖고 있어야 한다. 이를 이용하여 시스템 형상은 새롭게 만들어 진다.

3.3.2.3 변경관리(Change Management)

변경관리는 추상화 단계인 논리적 변경에 관한 정보를 제공하며, 새로운 버전이 생성될 때 중요한 역할을 한다. 모든 컴포넌트 버전은 이전 버전과의 차이점에 관한 정보를 갖고 있어야 한다. 그러나 이 정보는 자동으로 생성할 수 없기 때문에 개발자가 제공해야 한다. 이를 이용하여 컴포넌트 변경에 따른 시스템의 이상 유무를 판단할 수 있다. 시스템과 충돌 가능성은 앞에서 기술한 3가지 호환성 - 입출력 호환성, 인터페이스 호환성, 행위 호환성 -으로 판별할 수 있다.

3.3.2.4 의존관리(Dependency Management)

컴포넌트가 추가될 때는 물리적인 이진 컴포넌트(binary component)가 전달되기 때문에 컴포넌트 사이의 의존 관계 정보는 전달되지 않는다. 따라서 이 컴포넌트가 시스템에 어떤 영향을 미치는지에 관한 정보를 알아야 한다. 예를 들어 COM에서는 컴포넌트가 윈도우 레지스트리(Window Registry)를 통하여 참조되는 컴포넌트들을 찾는다. 윈도우 fp지스트리는 설치되는 모

든 컴포넌트들에 대한 interface id, class id, library location과 같은 데이터들이 저장되기 때문이다. 따라서 컴포넌트 사이의 관계는 처음 설치될 때 정해지고, 클라이언트(client)는 레지스트리에 있는 서버 컴포넌트를 키 값으로 찾게 되는 것이다. 그러면 COM은 클라이언트 메모리로 이와 관련된 컴포넌트를 로드시킨다. 의존관리는 시스템 전반에 걸친 모든 컴포넌트에 관련된 정보를 메타데이터(meta-data)에 의한 의존 그래프라는 인터페이스를 통하면 가능하다. 메타데이터는 새로운 컴포넌트 인터페이스로서 컴포넌트 등록 때 저장소에 저장된다.

4. 결 론

본 연구는 실시간 환경에서 컴포넌트 형상 관리의 필요성에 관하여 기술하였다. 컴포넌트 기반 시스템에 새로운 컴포넌트가 추가 또는 대체될 때 기존의 컴포넌트가 다른 패키지와 연결되어 있을 경우 변경과 의존관계의 불확실성, 또한 실시간 환경에서의 시스템에 대한 동적 행위에 관한 문제점 등을 분석하였다. 그리고 그 문제점의 해결방안인 컴포넌트 형상관리(CCM) 방법에 관하여 논하였다. 컴포넌트 변경은 라이브러리와 인터페이스로 나누어 설명하였고, 컴포넌트 형상관리는 기존의 SCM에 컴포넌트 관계성을 추가하여 버전관리, 형상과 빌드 관리, 변경관리, 의존관리 4가지로 구분하여 컴포넌트 관리기법을 기술하였다.

참고문헌

[1] Magnus Larsson, Ivica Crnkovic, "New Challenges for Configuration Management", Ninth International Symposium on System Configuration Management (SCM-9), Toulouse, France, September 1999.

[2] Ivica Crnkovic, "Component-based Software Engineering - New Challenges in Software Development", Invited talk & Invited report, MIPRO 2001 proceedings Opatija, Croatia , May 2001.

[3] George T. Heineman, William T. Councill, "Component-Based Software Engineering", Addison-Wesley, pp. 485-549, 2001.

[4] Alan W. Brown, K.C. Wallnau, " Engineering of Component-Based Systems", Proceedings of the 2nd IEEE International Conference on Complex Computer Systems, Oct. 1996.

[5] Magnus Larsson, Ivica Crnkovic, "Component Configuration Management", ECOOP 2000 Conference, Workshop on Component Oriented Programming, Nice, France, June, 2000.

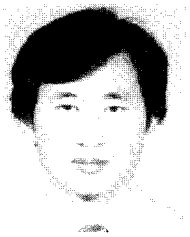
저자약력

한 정 수

1990년 경희대학교 전자계산공학과(공 학사)
1992년 경희대학교 전자계산공학과(공학석사)
2000년 경희대학교 전자계산공학과(공학박사)
2001년- 현재 천안대학교 정보통신학부 교수
관심분야 : CBSE, 컴포넌트 형상관리, EJB, S/W 재공학
이 메 일 : jshan@cheonan.ac.kr

고 응 남

1984년 연세대학교 수학과(이학사)
1991년 숭실대학교 전산공학과(공학석사)
2000년 성균관대학교 정보공학과(공학박사)
1983년 11월~1993년 2월 대우통신 컴퓨터 개발부 선임연구원
1993년 3월~1997년 2월 동우대학 전자계산과 교수 /전산실장
1997년 3월~2001년 2월 신성대학 컴퓨터 계열 교수/전산소장
2001년 3월- 현재 천안대학교 정보통신학부 교수
1999년 12월- 현재 멀티미디어 기술사
관심분야 : 멀티미디어, 인터넷, CSCW, 에이전트, 결함허용
이 메 일 : ssken@cheonan.ac.kr



이 정 배

1981년 경북대학교 전산공학과(공학사)
1983년 경북대학교 전산공학과(공학석사)
1995년 한양대학교 전산공학과(공학박사)
1982년~1991년 한국전자통신연구원 선임연구원
1996년~1997년 U.C.Irvine 객원교수 Dept.of Electrical & Computer Eng.
1991년- 현재 부산외국어대학교 컴퓨터공학과 부교수
관심분야:컴퓨터네트워크, 실시간시스템, 인터넷 응용
이 메 일 : jblee@sunmoon.ac.kr