범용성 향상을 위한 메시지 흐름 가변성 설계 및 특화 기법☆

A Variability Design and Customization Technique of Message Flow for Improving Generality

조 은 숙* 김 철 진**
Cho Eun Sook Kim Chul Jin

요 약

다양한 도메인의 요구사항을 만족시켜 주기 위한 컴포넌트는 내부에 다양성을 제공할 수 있도록 개발되어야 한다. 그러나컴포넌트 개발 시에 다양한 요구사항을 분석하여 설계하더라도 컴포넌트가이용될 때 또 다른 다양한 요구 사항들이 발생한다. 따라서 다양한 요구 사항들을 완전하게 만족시켜 주기 위한 컴포넌트의 설계는 매우 어려우며 또한 도메인의 특정화된 업무 로직을 완전하게 수용하는 것은 불가능하다. 이와 같은 문제들로 인해 컴포넌트가 블랙박스가 아닌 화이트 박스로 제공해야 하는 문제가 발생한다.

따라서 본 논문에서는 컴포넌트에 다양성을 제공하기 위해 다양한 도메인을 분석하여 컴포넌트를 설계하기 위한 기법보다는 다양한 도메인의 요구사항을 수용할 수 있는 장치를 제공하기 위해 컴포넌트의 가변성 설계 기법과 이런 설계 기법을 이용하여 컴포넌트를 특화(커스터마이제이션)하기 위한 기법을 제안한다. 컴포넌트의 가변성은 컴포넌트 개발(CD : Component Development) 과정에서 초기 가변성이 설계되며 가변성 적용을 위해 특화기법을 이용한다. 가변성이 적용된 컴포넌트를 이용하여 어플리케이션을 개발하는 과정에서 가변성이 재 설계될 수 있으며 이러한 과정을 통해 컴포넌트의 가변성이 진화되고 컴포넌트의 일반성이 더욱 향상될 수 있다. 일반적으로 컴포넌트의 가변성 범위는 컴포넌트 내부의 기능 변경과 컴포넌트 외부의 요구사항에 따라 컴포넌트 내부 구조가 변경되는 것으로 구분될 수 있다. 여기서는 이러한 가변성 범위에 따라 컴포넌트 내의 메시지 호출 변경을 위한 메시지 흐름(Message Flow) 설계 기법을 제안한다.

Abstract

The component for satisfying several domain requirements must be developed to support variety. But, when the application is developed using the component, it happens other requirements. So, it is difficult to design component to satisfy severaldomain requirements. Also, it is impossible to support the special business logic. As this problem, the component must provide to the white-box component, it is not the black-box component.

So, in this paper, we propose the variability design technique and the customization technique using the design technique that can support the various requirements. This technique is not focus on designing the component to analyze various domains. The variability of the component is designed to the initial variability in the component development phase and we use the customization technique for applying the variability to developing application. The variability can be re-designed during developing the application to use the component applying the variability. The variability of the component is evolved and the generation of the component is increased via the iteration Generally, the range of the component variability is classified the function modification within the component and the component internal structure modification as requirements in the component outside. As the range of the variability, we propose the variability design technique of the behavior and the message flow. This paper proposes a message flow design technique for modifying function call.

☞ keyword : Component, Component Interface, Message Flow, Variability, Customization, Component Development

1. 서 론

1.1 연구 배경

* 정 회 원 : 서일대학 소프트웨어과 조교수
escho@seoil.ac.kr
** 정 회 원 : 삼성전자 디지털 솔루션 센터
chuljin777.kim@samsung.com

[2007/07/31 투고 - 2007/08/17 심사 - 2008/01/16 심사완료]
☆ 본 논문은 2006년 서일대학 교내연구비에 의하여 지원되었음

다양한 도메인의 요구사항을 만족시켜 주기 위한 컴포넌트는 내부에 다양성을 제공할 수 있도록 개발되어야 한다. 그러나 컴포넌트 개발 시에 다양한 요구사항을 분석하여 설계하더라도 컴포넌트가 이용될 때 또 다른 다양한 요구 사항들이 발생한다. 따라서 다양한 요구 사항들을 완전하게 만족시켜 주기 위한 컴포넌트의 설계는 매우 어려우며 또한 도메인의 특정화된 업무 로직을 완전하게 수용하는 것은 불가능하다. 이와 같은 문제들로 인해 컴포넌트가 블랙박스가 아닌 화이트 박스로 제공해야 하는 문제가 발생한다.

또한 컴포넌트는 빠른 시간 내에 어플리케이션을 개발하기 위한 개발 블록(Building Block)으로 컴포넌트 이용자(Component User)들에게 컴포넌트 인터페이스(Component Interface)와 컴포넌트 스펙(Component Specification)을 제공한다[1,2,3]. 컴포넌트 인터페이스를 이용해다양한 도메인의 요구사항을 충족시키기 위해서는 컴포넌트 내부에 다양성을 제공해야 한다. 그러나이러한 다양성을 제공하기 위한 가변성을 설계하기가 어려우며 설계된 가변성을 적용하기 위한 기법들이 존재하지 않기 때문에 가변성 부분에 대해 컴포넌트를 공개하여 내부를 변경하는 실정이다. 따라서 본 논문에서는 컴포넌트의 가변성의 범위 가운데 메시지 흐름 가변성을 설계할 수 있는 기법을 제안하며 설계된 가변성을 기반으로 컴포넌트를 특화하기 위한 기법을 제안한다. 이는 컴포넌트를 이용하여 어플리케이션을 개발하는 과정에서 가변성이 재 설계될 수 있으며 이러한 과정을 통해 컴포넌트의 가변성이 진화되고 컴포넌트의 일반성이 더욱 향상될 수 있다.

본 논문의 2장에서 가변성 설계와 관련된 기존 연구들의 한계성들을 설명하고, 3장에서는 가변성 설계 프로세스를 제시한다. 4장에서는 메시지 흐름에 대한 가변성 설계 기법과 특화 기법을 제시하고, 5장에서는 사례 연구와 제시된 기법과 기존 기법들과의 비교 및 평가를 한다. 마지막으로 6장에서는 결론 및 향후 연구과제를 제시한다.

2. 기존 가변성 설계 기법

현재까지 소개된CBD(Component Based Development) 방법론은 컴포넌트 개발과 컴포넌트 기반 어플리케이션 개발이라는 2가지 측면을 고려하여 제시하고 있지만 컴포넌트 개발에만 집중하고 있는 상태다. 또한컴포넌트 개발 시점에 가변성에 대한 사항들을 정의하지 않고 있으며 이러한 가변성을 기반으로 컴포넌트를 이용하기 위한 절차를 정의하고 있지 않다. 몇 가지 CBD 방법론에서는 간헐적으로 가변성에 대한 언급을 하고 있다.

Catalysis[4]는 상호 연동할 수 있는 컴포넌트들을 가지고 어플리케이션을 개발하는 데 초점을 두고 있는데, 분석에서부터 구현까지 이르는 컴포넌트 모델링 기법들을 정의하며 컴포넌트들을 명세화하는 기법 등을 제시하고 있다. 그러나 각각의 단계에 정의된 절차들을 보면 다양한 도메인에 적용될 수 있도록 가변성을 추출하여 컴포넌트를설계하는 세부 지침을 정의하고 있지 않다. 부분적으로 컴포넌트의 가변성을 정의하기 위한 장치로 "Required Interface"를 정의하는 절차만 명시하고 있다. 그러나 이 인터페이스는 컴포넌트 내부에 존재하는 클래스들 간의 메시지 흐름을 변경할 수는 없고, 다만 외부에서 플러그-인 시킬 수 있는 장치로만 사용 가능하다. 그 결과 컴포넌트 내의 클래스들 간의 메시지 호출 경로 추가, 변경 및 삭제가 이루어질 수 없다. 또한 메시지호출에 대한 변경, 추가 및 삭제 메커니즘이 지원되고 있지 못하다.

Componentware에서는 컴포넌트 개발의 여러 단계들을 조합하여 다양한 형태의 프로세스를 구성할 수 있도록 프로세스 패턴들을 제공한다[5]. Componentware에서도 Catalysis 처럼 가변성 설계와 관련된 기법으로는 가변성 정의 장치인 "Import Interface" 정의 기법에 대해서만 제시하고 있다. 따라서 Catalysis와 동일하게 컴포넌트 내부 메시지 흐름에 대한 동적 제어 장치가 없기 때문에 메시지 호출 순서의 추가, 변경, 삭제가 불가

능하게 된다. 그리고, 컴포넌트는 하드웨어 컴포넌트처럼 재사용성이 높고 동적 바인딩이 용이하기 위해서는 블랙박스 개념의 컴포넌트 설계 기법이나 가변성 설계 지침이 구체적으로 반영되어 개발되어야 하는데 이 방법론에서는 단지 인터페이스 정의 수준에 머무르고 있다.

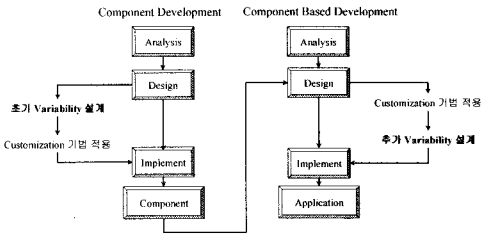
CoPAM(Component-Oriented Platform Architecting Method)[6]은 제품 공학(Product Engineering)을 위한 개발 프로세스로서, 플랫폼 엔지니어링(Platform Engineering)과 제품 엔지니어링 2개의 서브 프로세스로 구성되어 있으며, 플랫폼 공학에서는 여러 개의 재사용 컴포넌트들로 구성된 플랫폼을 개발하기 위한 프로세스를 정의하고 있으며, 제품 공학 프로세스에서는 이러한 플랫폼을 이용한 제품을 개발하기 위한 업무들을 정의하고 있다. 이 방법론은플랫폼 엔지니어링 프로세스에서 여러 도메인에 공통된 부분과 가변적인 부분들을 컴포넌트로추출하기 위한 활동에 초점을 두고 있다. 따라서, 컴포넌트 내부 설계 기법이나 컴포넌트 내부 메시지 흐름의 가변성 설계와 관련된 구체적인 지침은 제시하고 있지 않다. 다만, 가변성 메커니즘의 유형에 대해서만 언급하고 있다. 본 연구에서 컴포넌트 자체의 범용성과 재사용성 향상에 초점을 두고 제시하는 컴포넌트 내부 메시지 흐름의 동적 변경을 위한 설계 기법인 메시지 흐름 변경 기법, 메시지 흐름 가변성 설계 기법 등은 제시하고 있지 않다.

QADA(Quality-driven Architecture Design and quality Analysis)[7,9]은 고품질의 소프트웨어 아키텍처 개발을 위한 방법론으로서, 컴포넌트 기반 아키텍처 설계 기법에 초점을 두고 있다. 따라서, 여러 도메인에 재사용 가능한 컴포넌트가 되고 위한 요구사항 추출이나 추출된 컴포넌트들을 조립하는 아키텍처 설계 에 치중된 지침들로 구성되어 있다. 이 방법론에서는 컴포넌트 개념과 제품생산 기술을 연동한 아키텍처 설계 기법은 제시하지만, 컴포넌트 가변성 추출이나 설계 지침을 제시하고 있지 않다.

QASAR(Quality Attribute-oriented Software Architecture)[8]는 소프트웨어 품질에 대한 요구사항을 평가하고 아키텍처를 설계하는 기법으로서, 아키텍처 설계와 품질 속성들을 체계적으로 적용하는 기법들을 제시하고 있다. 그러나, 아키텍처 설계에서 가변성에 대한 설계와 품질 속성들을 가변성 설계에 어떻게 반영할 것인가에 대한 기법들은 제시하고 있지 않다. 특히 재사용성의 효과에 영향을 미치는 메시지 흐름 가변성과 관련한 설계 지침이나 메시지 흐름 호출 순서 제어 변경 또는 컴포넌트 결합도 축소 기법 등에 대한 기법들이 제시되지 않고 있다.

3. 가변성 설계 프로세스

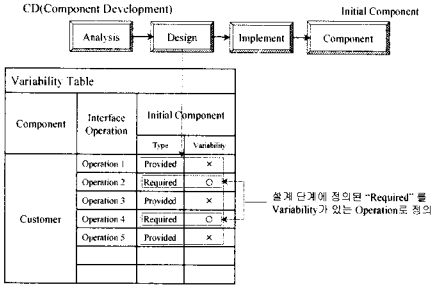
가변성을 설계하기 위한 프로세스는 (그림 1)과 같이 컴포넌트 개발 단계와 컴포넌트 기반 어플리케이션 개발 단계로 구분하여 설계될 수 있다. 이와 같이 컴포넌트 개발 단계에서는 초기 가변성을 설계하며 컴포넌트를 적용하여 어플리케이션을 개발하는 단계에서는 초기에 설계될 가변성에 대해 더욱 발전 시키며 추가적으로 새로운 가변성을 설계할 수 있다[10,11].



(그림 1) 가변성 설계 프로세스

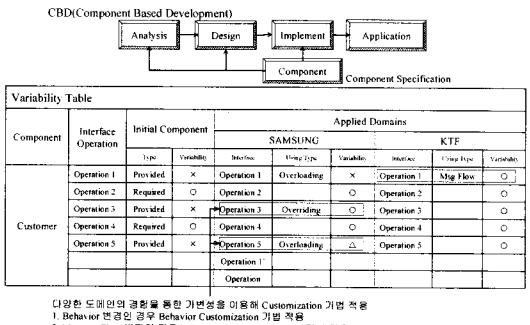
컴포넌트 개발 단계에 설계된 가변성은 어플리케이션 개발 시 특화 기법을 적용하여 변경될 수 있다. 제공된컴포넌트의 기능들 중에 가변성으로 제공된 기능 외에 다른 비가변성 기능들이 어플리케이션 개발 시에 변경을 요구할 수도 있으며 이러한 예외적인 변경 사항을 반영할 수 있도록

추가적으로 가변성을 설계할 수도 있다.



(그림 2) 컴포넌트 개발 단계의 가변성 추출

(그림 2)는 컴포넌트 개발 시점에 추출된 초기 가변성을 나타내며 가변성 테이블(Variability Table)에서 "Interface Operation" 중에 "type"이 "Required"로 표시된 기능을 초기 가변성으로 정의할 수 있다. 이와 같이 가변성 테이블에서 "Variability"가 ""로 표시된 "Interface Operation"에 대해 가변성 설계를 해야 한다. 초기 가변성은 컴포넌트 설계자(Component Designer)의 설계 범위에 따라 다양하게 적용될 수 있도록 가변성의 범위를 확대할 수도 있으며 단지 제안된 도메인에 사용될 수 있도록 한정적으로 가변성을 정의할 수도 있을 것이다. 이와 같이 초기 가변성은 설계자의 도메인 지식에 따라 좌우되기 때문에 컴포넌트를 이용하여 어플리케이션을 개발하는 과정에서 추가적으로 설계될 수 있다.



(그림 3) 컴포넌트 이용 단계의 가변성 추출

컴포넌트를 이용하여 도메인에 적용하는 경우에 가변성으로 정의되어 제공된 기능들은 특화 기법을 이용하여 적용될 수 있다. 그러나 가변성으로 정의되지 않은 기능들이 변경을 요구하는 경우에는 추가적으로 가변성을 설계해야 한다. (그림 3)의 가변성 테이블과 같이 기존 초기 가변성으로 설계된 기능 외의 기능들이 "오버라이딩(overriding)"이나 "오버로딩(overloading)", 또는 "메시지 흐름(Message Flow)"의 형태로 변경을 요구하는 경우에 추가적으로 가변성을 설계한다. 변경 사항은 표 1과 같이 정의할 수 있다.

(표 1) 컴포넌트 변경 사항

변경 사항	설 명
기능 추가	새로운 오퍼레이션 추가
오버라이딩	기존 오퍼레이션의 선언은 변경하지 않고 다른 기능으로 변경
오버로딩	기존 오퍼레이션의 함수 명은 동일하고 매개변수와 리턴 타입을 변경하여 새로운 기능으로 추가
메시지 흐름 변경	오퍼레이션 내의 메시지 흐름에 대한 변경

표 1에서의 컴포넌트 변경 사항 중에 오버라이딩과 메시지흐름 변경은 기존 기능에 대한 변경 사항이기 때문에 특화기법을 적용하여 제공되어야 할 가변성 범위이다. 그러나 "기능 추가"나 "오버로딩"은 기존 기능이 아니라 새로운 기능에 대한 추가이므로 가변성의 범위가 되지 않는다. 결과적으로 "기능 추가"나 "오버로딩"은 컴포넌트 개발 시에 기능을 고려하지 않고 설계되었기 때문에 도메인에 적용할 때 부족한 기능으로 판단되어 추가하는 결과가 발생하므로 컴포넌트설계가 완전하지 않았음을 나타낸다.

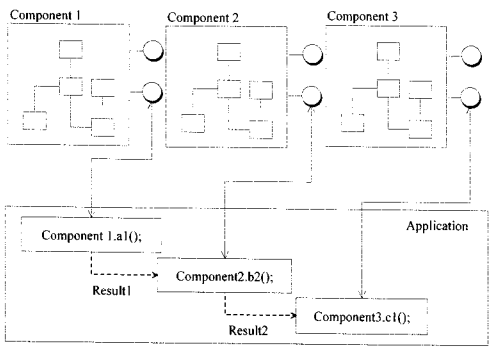
본 논문에서는 "기능 추가"나 "오버로딩"이 가변성의 범위는 아니지만 도메인에 적용되면서 컴포넌트의 일반성을 향상시키는 요소로 고려하여 초기 컴포넌트의 기능으로 추가할 수 있다. 따라서 추가된 기능도 변경 가능성을 판단하여 가변성을 설계한다.

4. 메시지 흐름 가변성 설계 및 특화 기법

본 논문에서는 컴포넌트 내의 메시지 흐름 가변성에 대해 가변성 설계 기법과 특화 기법을 제안한다. 이와 같은 가변성 설계 기법 및 특화 기법은 3장에서 제시된 가변성 설계 프로세스에 따라 컴포넌트 개발 단계와 컴포넌트를 이용한 어플리케이션 개발 단계에 모두 적용될 수 있다.

4.1 메시지 흐름

어플리케이션은 컴포넌트를 통합하여 개발되며 컴포넌트의 통합은 인터페이스의 메시지 흐름 조합을 의미한다. 이와 같이 메시지 흐름에 대한 조합은 어플리케이션에 따라 다양하게 변경될 수 있으며 또한 개발된 어플리케이션에서도 변경이 발생할 수 있다. 그러나 기존 개발 방법론들은 컴포넌트의 메시지 흐름에 대한 가변성을 고려하고 있지 않으며 메시지 흐름에 대한 강한 결합도 때문에 변경하기에 많은 부하가 걸린다[8].

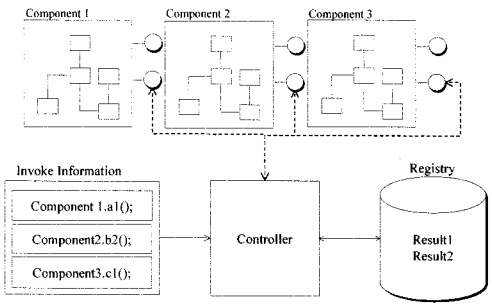


(그림 4) 기존 컴포넌트의 메시지 흐름

(그림 4)와 같이 비즈니스 어플리케이션을 개발하기위해 컴포넌트을 사용하여 프로세스를생성할 경우 컴포넌트들 간의 메시지 흐름이 정적으로 강하게 결합되어 있다. 이와 같이 강하게 결합되어 있는 비즈니스 프로세스 중에 하나의 컴포넌트 서비스를 삭제하거나 추가할 경우 프로세스 내의 다른 컴포넌트에 많은 영향을 준다. (그림 4)에서 비즈니스 프로세스는 "Component1", "Component2", "Componnet3"를 호출하고 있으며 각각의 컴포넌트의 처리 결과를 다음 컴포넌트의 입력 파라메터로 이용하고 있다. 만약 "Component2" 가 다른 컴포넌트로 대치 되거나 삭제되는 경우 "Component1"은 어떤 컴포넌트에 결과 값을 전달해야 할지, 그리고"Component3"는 어떤 컴포넌트의 결과 값을 입력으로 받아야 할지 변경이 복잡해 진다.

4.2 메시지 흐름 가변성 설계 기법

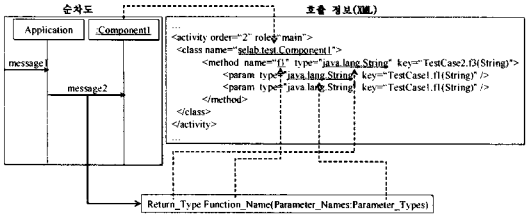
본 논문에서는 어플리케이션 개발 시 컴포넌트 간의 강한 결합도를 줄이기 위해(그림 5)와 같이 컴포넌트의 메시지 흐름을 중앙 집중식으로 제어할 수 있도록 "Controller"라는 개념을 적용하여 메시지 흐름 가변성 설계 기법을 제안한다. 기본 메커니즘은 "Controller"가 호출 정보를 참조하여 컴포넌트를 호출하고 결과값은 레지스트리(Registry)에 저장하여 다음 메시지 호출 시 참조할 수 있도록 한다. 이런 기법으로 설계가 되면 컴포넌트의 메시지 흐름에 대한 변경은 단지 호출 정보만 변경하면 다른 부분에 전혀 영향을 주지 않고 동적으로 변경이 가능하다.



(그림 5) 동적 컴포넌트 메시지 흐름

(그림 5)와 같이 "Controller"는 호출 정보(Invoke Information)를 통해 컴포넌트의 메시지를 호출하

며 결과값은 "Registry"에 저장한다. 이러한 호출 정보는 XML로 저장되며 이 호출 정보가 메시지 흐름 가변성을 위한 설계의 핵심이 된다. 호출 정보의설계는 순차도(Sequence Diagram)의 정보를 이용할 수 있으며 각각의 메시지 정보는 XML로 생성 된다.



(그림 6) 순차도를 통한 호출 정보 생성

(그림 6)은 어플리케이션 개발 시 컴포넌트들 간의 메시지 흐름을 나타내는 순차도와 이러한 순차도를 이용해 생성된 호출 정보를 보여준다. 순차도의 메시지 정보는 XML 호출 정보의 태그(Tag) 속성 데이터로 매핑되어 생성된다. 메시지 정보와 호출 정보의 매핑 관계는 표 2와 같다. 이렇게 생성되는 호출 정보는 메시지 정보 중에서 필요한 데이터 만을 매핑 시키며 또한 부수적으로 필요한 정보는 추가하여 생성한다. 예를 들면 메시지 정보 중에 "Parameter_Names"는 메시지를 호출할 때 "Parameter_Types"를 통해 함수를 호출하기 때문에 파라메터 명은 필요하지 않으므로 호출 정보에 포함시키지 않는다. 또한 호출 되는 메시지의 처리 결과값을 레지스트리에 저장하도록 하기 위해 키를 생성해서 호출 정보에 추가한다.

(표 2) 메시지 정보와 호출 정보의Mapping

메시지 정보	호출 정보
클래스 이름	<class name="?">
리턴 타입	<method type="?">
함수 이름	<method name="?">
Parameter_Names	None
Parameter_Types	<param type="?">

XML호출 정보에서 "<Activity>" 태그는 호출하려고 하는 하나의 컴포넌트의 호출 정보를 담고 있으며 호출하는 컴포넌트 메시지 개수에 해당하는 "<Activity>"가 생성된다. 각각의 "<Activity>"는 "Order" 속성을 가지고 있으며 다음 메시지로 진행하기 위한 역할을 한다. 각각의 "<Activity>" 태그 내의 컴포넌트 호출 정보에 대한 정의는 표 3와 같다.

(표 3) 호출 정보의태그 정의

정보		태그	속성	타입	설명
컴포넌트	컴포넌트 이름	<class>	Name	String	Full Package Name
컴포넌트 인터페이스	인터페이스 이름	<method>	Name	String	
	리턴 타입	<method>	Type	String	Full Package Name
	Saving Key	<method>	Key	String	To Data Registry
Parameter (Multiple)	Parameter Type	<param>	Type	String	Full Package Name
	Loading Key	<param>	Key	String	From Data Registry

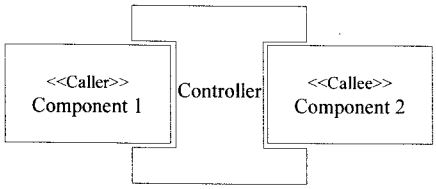
위의 표에서와 같이 컴포넌트를 호출하기 위한 정보는 컴포넌트 명, 인터페이스 관련 정보, 파라메터 관련 정보로 구성된다. 인터페이스의 "Saving Key"나 인터페이스의 "Loading Key"는 레지스트리로 처리 데이터를 저장하거나 호출하기 위해 사용되는 키이다.

이러한 메시지 호출 정보를 XML로 설계하는 이유는 메시지 흐름에 대한 다양한 요구 사항을 쉽고 동적으로 변경할 수 있기 때문이다. 메시지 흐름 가변성을 위한 지금까지의 설계 기법은 특화 기법을 통해 어떻게 변경되는지 파악할 수 있다.

4.3 메시지 흐름 특화 기법

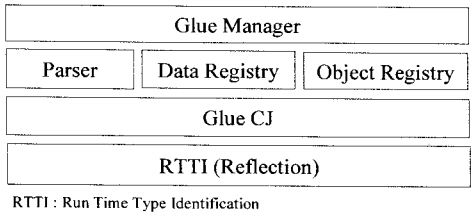
메시지 흐름 가변성 설계를 통해 생성된 호출

정보는 컴포넌트의 호출을 동적으로 변경할 수 있도록 하기 위한 데이터로 제공된다. 물론 (그림 5)에서 호출 정보인 XML을 해석하고 처리하기 위한 "Controller"라는 엔진을 고려해야 한다. 본논문에서 제안하는 컴포넌트 메시지 흐름 특화 기법은 (그림 7) 에서와 같이 'Controller'라는 중간 매개체를 통해 컴포넌트들은 서로 호출하고 호출될 수 있다(Glue Mechanism). 호출하고 호출되는 컴포넌트들은 서로의 호출 정보를 가지고 있지 않으며 단지 "Controller"가 호출 정보를 가지고 동적으로 흐름을 제어한다. 가변성 설계를 통해 생성된 호출 정보가 "Controller"의 입력 데이터로 이용되어 메시지들을 실행한다.



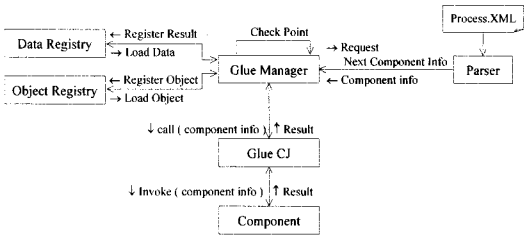
(그림 7) Glue 개념

(그림 7) 에서와 같이 호출하는 "Component 1"은 호출되는 "Component 2"의 인터페이스 정보를 전혀 가지고 있지 않으며 "Component 2"가 다른 컴포넌트로 교체되더라도"Component 1"에 전혀 영향을 미치지 않는다. 단지 'Controller'에 존재하는 컴포넌트 호출 정보에 컴포넌트의 교체 정보만 수정하면 "Component 1"은 교체된 다른 컴포넌트를 호출하도록 한다.



(그림 8) Controller Layer Architecture

"Controller"의 아키텍쳐는 (그림 8)에서 보는 것과 같이 동적으로 컴포넌트 호출할 수 있도록 동적 타입 정의 기능인 리플렉션(Reflection)을 기반으로 하고 있으며[9,10], 'GlueCJ'는 이러한 리플렉션 기능을 이용한다. 어플리케이션에서 정의된 프로세스를 이용하기 위해 제어 역할을 하는 것은 'GlueManager'가 담당한다. 'GlueManager'는 입력 받은 프로세스 XML정보를 'Parser'를 통해 해석하여 'GlueCJ'로 전달한다.



(그림 9) Controller Flow 아키텍처

(그림 9)와 같이 'GlueCJ'는 'GlueManager'로부터 호출 정보에 관련된 XML를 얻어 컴포넌트의 기능을 처리한 후 결과를 'GlueManager'로 전달한다. 'GlueManager'로부터 XML 정보를 얻은 'GlueCJ'는 컴포넌트를 호출하기 위해 다음과 같은 Interface를 이용한다.

```
public Object call(String componentName, String methodName, Object[] methodParams)
```

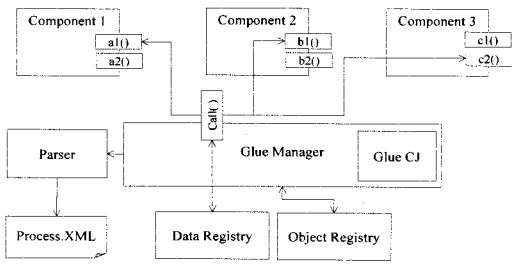
호출 정보인 XML로부터 해석된 정보 중에 컴포넌트 명과 함수 명을 얻을 수 있으며 함수의 파라메터 데이터는 'Data Registry'로부터 읽어와 객체 배열을 형성하여 'GlueCJ'의 'call()' 함수를 호출한다. 컴포넌트 호출 처리 결과는 'Data Registry'에 저장되며 또한 'Object Registry'는 한번 호출된 컴포넌트의 객체를 캐싱하기 위해 이용된다.

```
public Object call(String componentName, String methodName, Object[] methodParams)
{
  Class _class                = null;
  Class _retType              = null;
  Method _method              = null;
  Object _object              = null;
  Class[] _methodParamsType   = null;
  Object result               = null;
  try
  {
    _class                    = Class.forName(componentName);   ①
    _methodParamsType         = getClass(methodParams);         ②
    _method                   = _class.getMethod(methodName, methodParamsType ); ③
    _object                   = _class.newInstance(); ④
    result                    = _method.invoke(_object, methodParams); ⑤
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
  return result;
}
```

(그림 10) GlueCJ의 Call 함수

(그림 10)은 'GlueCJ'의 'call()'함수로서 컴포넌트의 정보를 받아 리플렉션 기능을 통해 해당 컴포넌트를 호출한다. 컴포넌트 명을 통해 컴포넌트 클래스를 얻으며() 이 클래스를 통해 객체를 생성한다(). 입력된 함수 파라메터 데이터에 대한 타입(type)을 얻은() 다음 함수 명과 함수 파라메터를 통해 함수 객체를 생성한다(). 생성된 함수 객체와 컴포넌트 객체를 이용하여 컴포넌트 서비스를 호출한다(). 이러한 컴포넌트 호출 메커니즘은 프로세스 정보를 담고 있는 XML를 읽어 반복적으로 처리된다.

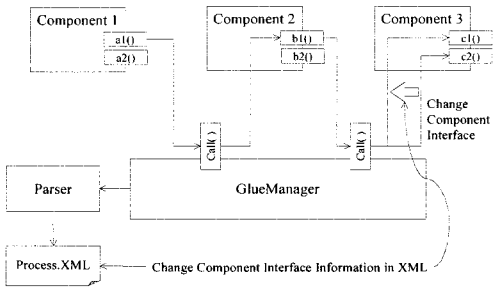
'GlueManager'는 처리된 결과를 다른 컴포넌트에서 이용될 수 있도록 'Data Registry'에 저장한다. 'Data Registry'는 'Glue Manager' 객체와 라이프 타임(Life Time)이 같기 때문에 프로세스 마다 독자적인 'Data Registry'를 가지고 있다. 아키텍처에서 'Object Registry'는 'GlueCJ'를 통해 호출된 객체를 다시 생성하지 않고 'Object Registry'에 등록하여 재 요청 시 호출하여 사용한다.



(그림 11) Glue Manager의 Call Mechanism

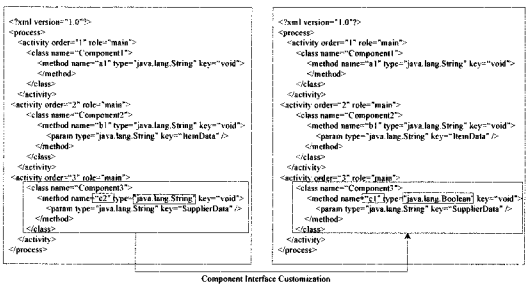
컴포넌트 호출과 처리 메커니즘은 (그림 11)에서 보는 바와 같이 호출하는 모든 컴포넌트는 "GlueManager"의 "call()" 함수를 호출한다. "Glue Manager"는컴포넌트의 흐름 정보를 가지고 있는 XML 정보를 읽어 컴포넌트를 반복적으로 호출한다. 이때 "<Activity>" 태그의 "order" 값을 이용하여 다음 컴포넌트를 호출한다. 처리 결과는 'Data Registry'에 저장되며 다른 컴포넌트 호출 시 입력으로 필요한 데이터를 읽어와 이용한다.

메시지 흐름 가변성의 종류는 컴포넌트 인터페이스 변경, 컴포넌트 추가, 그리고 컴포넌트 삭제로 구분된다.



(그림 12) 컴포넌트 Interface 변경

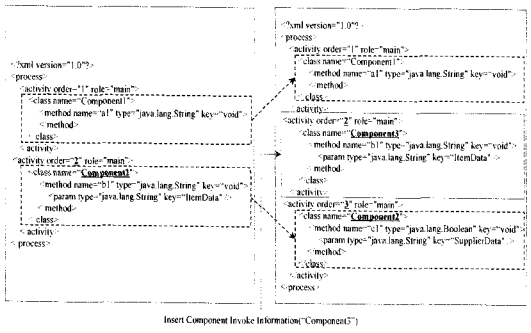
(그림 12)는 컴포넌트 인터페이스를 변경하는 메커니즘으로 호출되는 컴포넌트의 오버로딩된 다른 인터페이스로 변경하는 과정을 보여주고 있다. 변경 전의 흐름은 "Component2"의 "b1()" 인터페이스가 "Component3"의 인터페이스 "c2()"를 호출하기 위해 "GlueManager"의 "call()"을 호출하고 있다.



(그림 13) 컴포넌트 Interface 변경 코드

(그림 13)과 같이 컴포넌트의 인터페이스를 변경하기 위해 기존 흐름인 "Component3"의 "c2()"로의 호출을 "Component3의 "c1()"으로 "Process.XML" 정보만을 변경하면 동적으로 컴포넌트의 흐름을 변경할 수 있다.

컴포넌트 인터페이스의 변경 메커니즘과 유사하게 컴포넌트의 추가 메커니즘도 "Process.XML" 정보를 변경하여 동적으로 컴포넌트를추가할 수 있다. (그림 14)와 같이 "Component 1"에서 "Component 2"로 메시지 흐름 사이에 "Component 3"를 쉽게 추가할 수 있다. 추가 할 때 삽입되는 컴포넌트의 호출 정보에 대해 "<Activity>" 태그의 "order"를 "Component2"는 기존에 "2"에서 "3"으로 변경해야 하며 추가된 "Component3"의 "order"는 "2"로 설정한다. 이와 같이 "Controller"가 순서적으로 컴포넌트의 메시지를 호출할 수 있도록 호출 정보를 변경한다.



(그림 14) 컴포넌트 추가

컴포넌트의 삭제 메커니즘은 기존 흐름에서 컴포넌트를 삭제하는 경우로서 다른 예외 사항들이 발생할 수 있다. 단순하게 "Process.XML"에서 삭제하려는 컴포넌트정보만을 삭제하면 되는 것이 아니라, 삭제 컴포넌트 전후에 관련된 데이터가 존재하기 때문에 이러한 데이터들에 대한 처리가 필요하다. 그렇지 않으며 컴포넌트의 삭제와 동시에 그 프로세스는 쓸모 없게 된다.

지금까지 메시지 흐름 가변성 설계 기법과 그 설계 데이터를 이용한 특화기법을 알아보았다. 메시지 흐름 가변성에 대한 설계 및 특화기법은 설계 단계뿐만 아니라 구현 단계까지 고려하여 동적으로 변경될 수 있음을 보였다. 이런 식의 동적인 메시지의 흐름 변경을 통해 어플리케이션 개발의 효율성을 증가시킬 수 있다.

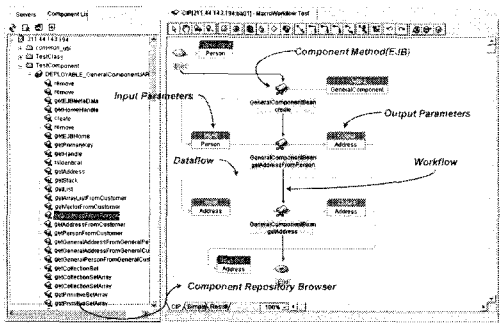
5. 사례 연구 및 평가

5.1 사례 연구

본 논문에서 제시하는 메시지 흐름 가변성 설계 기법을 검증하기 위해 본 사례 연구에서는 가변성 설계 기법의 핵심 프레임워크인 Controller를 구현한 사례를 통해 증명한다.

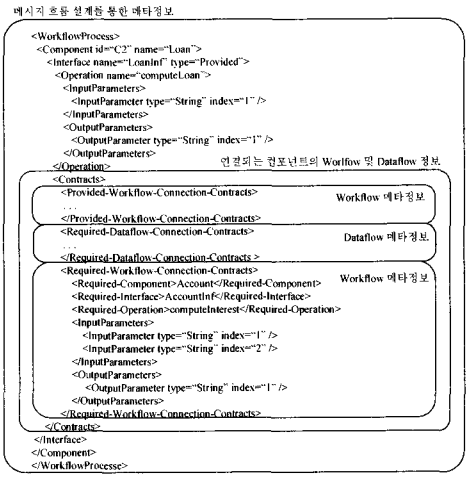
(그림 15)는 Controller의 메커니즘을 이용하여 구현한 컴포넌트 통합 도구 이다. 컴포넌트들 간의 통합을 통해 메시지 흐름을 구성하기 위해서는 왼쪽의 컴포넌트 Repository Browser로부터 컴포넌트 선택하여 오른쪽 설계 창에 설계한다. 하나의 컴포넌트는 다른 컴포넌트의 기능을 호출하기 위해서 워크플로우를 설계하며, 메시지 간의 데이터 흐름은 하나의 컴포넌트 출력(Component's Output Parameter)이 다른 컴포넌트의 입력(Component's Input Parameter)으로 데이터 흐름을 설계한다.

워크 플로우와 데이터 흐름 설계는 컴포넌트들 간의 메시지 흐름 설계의 기본이 되며 워크 플로우와 데이터 흐름의 변경을 통해 동적으로 컴포넌트의 흐름을 특화시킬 수 있다. 본 메시지 흐름 설계 도구는 (그림 9)의 아키텍처에서와 같이 동적으로 물리적인 컴포넌트를 호출하여 실행할 수 있으며 동적으로 컴포넌트를 교체하거나 삭제하여 메시지 흐름을 동적으로 관리할 수 있다. 본 메시지 흐름 설계 도구는 컴포넌트 통합 설계뿐만 아니라 운영 시에 이용될 수 있으며 동적인 관리가 가능하다.



(그림 15) 메시지 흐름 설계 도구

본 도구에서는 워크 플로우와 데이터 흐름의 설계 후에 자동으로 메타정보가 생성되며 XML 형태로 정의 되어 동적으로 관리될 수 있다. (그림 16)과 같이 워크플로우 메타정보와 데이터 흐름 메타정보로 구성되며 외부에 제공될 정보는 Provided- 태그에 정의하고 있으며 외부에 요구되는 워크 플로우 및 데이터흐름에 대한 정보는 Required- 태그에 정의한다.



(그림 16) 메시지 흐름 메타 정보

XML 형태의 메타정보는 실제적으로 Controller를 통해 컴포넌트들의 오퍼레이션을 호출하기 위한 정보들을 포함하고 있다. 또한 처리된결과 데이터를 통해 다른 오퍼레이션의 입력 데이터로 이용될 수 있도록 매핑 정보도 정의하고 있다. 컴

(표 4) 가변성 특징및 기능 비교

비교 항목			제안기법	Catalysis	Componentware	CoPAM	QADA	QASAR
특징	가변성 추출 기법 여부		N	P	P	P	N	N
	가변성 설계 지침 여부		S	N	N	S	N	N
	블랙박스 설계 지원 여부		S	N	N	N	N	N
	동적 특화 기법 지원 여부		S	N	N	N	N	N
	컴포넌트 결합도 축소 지원 여부		S	N	N	N	N	N
기능	Attribute	속성 변경	N	N	N	N	N	N
	Behavior	Interface 변경	S	N	N	S	P	P
		Interface 추가	S	N	N	S	P	P
		컴포넌트내의 Class 교체	S	N	N	N	N	N
	Message Flow	메시지 호출 변경	S	N	N	N	N	N
		메시지 호출 추가	S	N	N	N	N	N
		메시지 호출 삭제	P	N	N	N	N	N

[S] Support [N] Not Support [P] Partial Support

포넌트 통합은 이러한 메타 정보를 통해 운영될 수 있도록 Controller를 기반으로 한다.

지금까지 본 논문에서 제시한 가변성 설계 기법의 핵심 메커니즘인 Controller를 통해 구현된 메시지 흐름 설계 도구를 살펴보았다. 본 설계 도구에서와 같이 Controller를 통해 가변적인 컴포넌트 통합 설계가 가능하며 동적인 커스터마이제이션을 통해 재사용성 높은 통합 컴포넌트(메시지 흐름) 개발이 가능함을 파악할 수 있다.

5.2 가변성 특징 및 기능 평가

본 논문에서 제안한 가변성 설계 기법 및 특화 기법이 다른 컴포넌트 개발 방법론에서 제안한 기법들과 어떤 차이가 있는지 비교 평가한다. 표 4에서 보는 것처럼 다른 방법론에서는 가변성에 대한 설계 기법이나 특화 기법들을 언급하고 있지 않다. 특히 기존 방법론들에서는 가변성 설계나 특화 기법에 대한 지원이 제시되지 않고 있으며, 컴포넌트의 행위 변경은 일부 지원되고 있지만 메시지 흐름에 대한 변경 기법은 지원되지 않고 있다.

본 논문에서는 가변성 추출과 관련된 기법은 다루고 있지 않고, 단지 분석 단계에서 추출된 가변성에 대해 설계하기 위한 기법들을 제안하고 있다. 그러나, 기존의 다른 기법들에서 언급하지 않고 있는 행위나 메시지 흐름에 대한 가변성을 지원하고 있다.

5.3 가변성 설계 기술 평가

가변성 설계를 하는데 있어서는 다양한 기술 혹은 기법들이 요구되어 사용되고 있다. 표5에 나타난 바와 같이 가변성 설계에는 포함, 상속, 매개화, 동적 클래스 로딩 등과 같은 다양한 기술들이 반영되어야 한다. 표5는 이러한 다양한 가변성 설계 기술들이 FODA, FAST, Kobra 등과 같은 기존 방법론들과 본 논문에 제시한 기법에서 얼마

(표 5) 가변성 설계 기술 평가

방법론 / 설계 기술	KobrA	FAST	FODA	COPAM	Catalysis	QADA	QASAR	COMO	Mic00	Dir03	제안기법
Aggregation (Delegation)			S		S	M			M		S
Inheritance		S	S		M			S	M		S
Parameterization		S	S			M		S	M	M	S
Dynamic Class Loading									M		
Plug-In				M		M	M	S			S
Templates					M	M		S			
오버로딩 (Polymorphism)					M				M	M	S
Dynamic Class Loading											S
Static Library									M		
DLL (Dynamic Link Library)									M		
Conditional Compilation		S							M	M	
Reflection									M		
AOP (Aspect-Oriented Prog.)									M	M	
Connector					M	M					S
Adaptor											S
Wrapper											
Mediator											S
Proxy											S
Re-implementation											

S: Supported, M: Mentioned Only, Blank: Not Supported

나 지원되고 있는지 평가한 결과를 제시하고 있다[12,13,14,15,16]. 표5에 나타난 바와 같이 기존 방법론들에서는 필요성에 대한 언급이나 정의는 하고 있지만 구체적인 기법들을 제시하고 있지 못하다. 이에 비해 본 연구에서는 실제 프로젝트 적용에 필요한 지침들을 구체적으로 제시하고 있음을 평가표를 통해 알 수 있다. 특히, 커넥터, 어댑터, 중개자, 프락시 등과같은 개념들은 본 기법에서만 제안하고 있음을 알 수 있으며, 보편적으로 기존 기법들에서는 상속이나 매개화, 동적 클래스 로딩 등과 같은 개념 정도만 지원하고 있다.

6. 결론

지금까지 본 논문에서는 컴포넌트의 메시지 흐름에 대한 가변성 설계 및 특화 기법을 제안했다. 지금까지 이루어진 여러 가변성 설계 기법들에서는 가변성에 대한 특징들을 가운데 일부 추출 및 설계 기법만 제시한 반면에 본 연구에서는 동적으로 컴포넌트를 특화할 수 있는 메시지 흐름에 대한 가변성 설계 기법을 제시하였다. 이는 순차도에 설계된 메시지의 정보를 XML로 매핑하여 동적으로 특화할 수 있는 기법으로서 컴포넌트기반 애플리 케이션 개발 시 요구되는 다양한 요구사항을 반영할 수 있도록 하기 위함이다. 본 논문에서제안한 기법은 기존 컴포넌트를 이용해 소프트웨어를 개발할 때 컴포넌트들 간의 강한 결합도를 최소화할 수 있으며 컴포넌트를 블랙 박스로 이용할 수 있도록 해준다. 또한본 연구에서 제시한 가변성 설계 기법을 통해 다양한 요구 사항을 동적으로 수용하여 반영할 수 있도록 하여 컴포넌트의 범용성을 더욱 향상시킬 수 있도록 하였다.

향후 연구과제로는 메시지 흐름 가변성에서 제안한 일반 직렬적인(Serial) 메시지 흐름뿐만 아니라 조건 분기와 같은 논리적이 변경도 지원할 수 있는 설계 기법이 있다. 또한 메시지 흐름에서 특정 컴포넌트의 메시지를 삭제했을 때 관련된 컴포넌트들의 영향 분석하여 안전하게 변경될 수 있는 설계기법과 특화 기법을 연구한다.

참고 문헌

[1] Szyperski C., Component Software: Beyond Object-Oriented Programming, Addison Wesley Longman, Reading, Mass., 1998.

[2] Kang K, "Issues in Component-Based Software Engineering", 1999 International Workshop on Component-Based Software Engineering.

[3] Hopkins J., "Component Primer", Communication of the ACM Vol. 43, No.10 , October 2000.

[4] D'souza D. F. and Wills A. C., Objects, Components, and Components with UML, Addison-Wesley, 1998.

[5] Rausch A. "Software Evolution in COMPONENTWARE Using Requirements/Assurances Contracts", Proceedings of the 22th International Conference on Software Engineering, 06/2000.

[6] P. America, H. Obbink, R. V. Ommering, F. V. D. Linden, "CoPAM: A Component-Oriented Platform Architecting Method Family for Product Family Engineering", The First Software Product Line Conference(SPLC), Kluwer International Series in Software Engineering and Computer Science, Denver, Colorado, USA, p.15, 2000.

[7] Matinlassi, Mari, Niemelä, Eila & Dobrica, Liliana, "Quality-driven Architecture Design and Quality Analysis Method: A revolutionary initiation approach to a product line architecture", VTT publications 456, VTT Technical Research Center of Finland, (URL:http://www.inf.vtt.fi/inf/pdf/), January 2002.

[8] E. Folmer, J. V. Gurp, and J. Bosch, "Scenarion-Based Assessment of Software Architecture Usability", In the Proceedings of Workshop on Bridging the Gaps Between

Software Engineering and Human-Computer Interaction, ICSE, Portland, 2003.

[9] Matinlassi, M., "Comparison of software product line architecture design methods: COPA, FAST, FORM, KobrA and QADA", In the proceedings of the 29th International Conference on Software Engineering, IEEE Computer Society, Washington Brussels Tokyo, Scotland, UK, May 26th - 28th2004. pp. 127 - 136.

[10] Bachmann F., Bass Len, http://www.sei.cmu.edu/plp/variability.pdf, "Managing Variability in Software Architecture", Software Engineering Institute(SEI), 2001.

[11] Chul Jin Kim, Soo Dong Kim, "A Component Workflow Customization Technique", Journal of KISS: Software and Applications, Volume 27, Number 5, May 2000.

[12] Kang, K. C., Cohen, S. G., Novak, W. E. and Peterson, A. S., "Feature-oriented Domain Analysis (FODA) Feasibility Study, " Technical Report CMU/SEI-90-TR-21, Software Engineering Institute (SEI), November 1990.

[13] Atkinson C., Bayer J., Bunse C., Kamstices E., Laitenberger O., Laqua R., Muthig D., Paech B., Wust J., and Zettel J., Component-based Product Line Engineering with UML, Addison-Wesley, 2001.

[14] Fowler M., and Scott K., UML Distilled: Applying the Standard Object Modeling Language, Addison-Wesley, 1998.

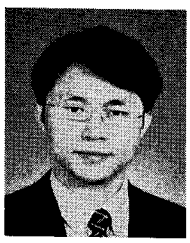
[15] Coplien J., Hoffman D., and Weiss D., "Commonality and Variability in Software Engineering", IEEE Software, pp. 37-45, November 1998.

[16] Weiss D. M., "Commonality Analysis: A Systematic Process for Defining Families," Second International Workshop on Development and Evolution of Software Architectures for Product Families, February 1998.

◐ 저 자 소 개 ◑

조 은 숙 (Cho Eun Sook)

1993년 동의대학교 전산통계학과 졸업(학사)
1996년 숭실대학교 대학원 컴퓨터학과 졸업(석사)
2000년 숭실대학교 대학원 컴퓨터학과 졸업(박사)
2002년~2003년 한국전자통신연구원 초빙연구원
2000년~2004년 동덕여자대학교 정보학부 전임강사
2005년 ~ 현재 서일대학 소프트웨어과 조교수
관심분야 : 컴포넌트 기반 소프트웨어 개발, 임베디드 소프트웨어개발, 유비쿼터스 컴퓨팅
E-mail : escho@seoil.ac.kr



김 철 진 (Kim Chul Jin)

1996년 경기대학교 전자계산학과 졸업(학사)
1998년 숭실대학교 대학원 컴퓨터학과 졸업(석사)
2004년 숭실대학교 대학원 컴퓨터학과 졸업(박사)
2004년~2005 가톨릭대학교 컴퓨터 정보 공학부 교수
2005 ~ 현재 삼성전자 디지털 솔루션 센터
관심분야 : 컴포넌트 기반 소프트웨어 공학, 임베디드 소프트웨어 개발 방법론
E-mail : cjkim777@gmail.com