MDA기반 이동 단말 시스템 소프트웨어 개발 기법

이 준 상[†] · 채 흥 석[††]

요 약

현대의 소프트웨어공학 관련 연구 중에서, 산업계가 궁극적으로 추구하는 수준의 생산성을 제공할 수 있는 기술은 아마도 프로덕트라인 공학이 될 것이다. 지금까지의 소프트웨어공학 기술로는 소프트웨어 개발 분야에 프로덕트라인 기법을 실질적이고 실용적으로 적용하기에는 아직 충분히 성숙하지 못한 것이 사실이다. 본 논문에서는 저자가 산업체에서 접한 과거 3년 동안의 PDA 스마트폰 개발 경험을 바탕으로 실용적 수준의 프로덕트라인 기법을 Model-Driven Architecture(MDA) 접근 방법을 통해 제안한다. 이동 단말 시스템의 경우 단말기 제조사, 목적 사용자 층, 이동 단말 사업자 등에 따라 다양한 형태의 소프트웨어가 존재한다. 특히, 최근에는 단말기 사용자가 직접 느낄 수 있는 지원 기능 구성 및 인터페이스 형식은 매우 다양한 형태가 존재하며, 같은 제조사에 시리즈 제품으로 개발하더라도 이 부분에 대한 변이성이 매우 큰 특징이 있다. 하지만, 전형적인 폰 관련 기능 모듈 즉, 음성호 및 화상전화 기능, 메시지, 주소록, 데이터통신, 카메라 및 멀티미디어 기능, 웹 브라우징과 같은 큰 기능 묶음에 대한 피처(Feature)들은 자체의 다양한 기능적 요구사항과 함께 피처 간 상호 연관성을 크고 다양한 형태로 구성될 수 있는 특성이 있다. 본 논문에서는 이동 단말기에서 구현해야 하는 다양한 형태의 사용자 소프트웨어의 요구사항에 대해 사용 시나리오 상에서 구분 가능하고 의미 있는 장면의 연속 관계로 정의하여 추후 설계, 구현, 시험 단계에서도 소프트웨어 아키텍처 역할을 할 수 있는 개발 기법을 제안한다. 따라서, 요구분석 단계에서도 사용자 인터페이스 관점에서 전반적인 소프트웨어 아키텍처에 대해 검증할 수 있게 될 뿐만 아니라 소프트웨어 개발 주기 전 과정에서 그 구조를 사용자 인터페이스 관점에서 유지, 관리 할 수 있는 핵심적인 방법을 제공한다.

키워드 : 소프트웨어공학, 소프트웨어 프로덕트라인, 이동 단말 시스템, MDA

A MDA-based Approach to Developing UI Architecture for Mobile Telephony Software

Joon-Sang Lee[†] · Heung Seok Chae[††]

ABSTRACT

Product-line engineering is a dreaming goal in software engineering research. Unfortunately, the current underlying technologies do not seem to be still not much matured enough to make it viable in the industry. Based on our experiences in working on mobile telephony systems over 3 years, now we are in the course of developing an approach to product-line engineering for mobile telephony system software. In this paper, the experiences are shared together with our research motivation and idea. Consequently, we propose an approach to building and maintaining telephony application logics from the perspective of scenes. As a Domain-Specific Language(DSL), Menu Navigation Viewpoint(MNV) DSL is designed to deal with the problem domain of telephony applications. The functional requirements on how a set of telephony application logics are configured can be so various depending on manufacturer, product concept, service carrier, and so on. However, there is a commonality that all of the currently used telephony application logics can be generally described from the point of user's view, with a set of functional features that can be combinatorially synthesized from typical telephony services(i.e. voice/video telephony, CBS/SMS/MMS, address book, data connection, camera/multimedia, web browsing, etc.), and their possible connectivity. MNV DSL description acts as a backbone software architecture based on which the other types of telephony application logics are placed and aligned to work together globally.

Key Words : Software Engineering, Software Product-line, Mobile Telephony Systems, MDA

1. 서 론

일반 가전이나 휴대용 전자기기 등에 다양한 기능과 네트워크 서비스를 제공하기 위한 목적으로 인해 최근 10년 동안 임베디드 소프트웨어에 대한 요구가 크게 증대되었다. 컴퓨팅 기능과 네트워크 성능이 향상되면서 임베디드 소프트웨어의 응용 분야가 우리 생활 깊게 영향을 미치고 있는 상황이다. 이러한 경향은 이동 단말 시스템에서도 예외가 아닌데, 특히 제공되는 기능상의 피처(Feature)나 성능측면에서 매우 급격한 발전을 이루고 있으며 이에 반에 개발착

※ 이 논문은 교육인적자원부 지방연구중심대학육성사업(차세대물류IT기술연구사업단)의 지원에 의하여 연구되었음.
† 정 회 원 : LG전자 디지털 미디어 연구소 선임연구원
†† 정 회 원 : 부산대학교 공과대학 컴퓨터공학과
논문접수 : 2006년 4월 4일, 심사완료 : 2006년 4월 27일

수 후 제품화 하기까지의 시간(lead time)은 점점 짧아지고 있어 시장으로부터의 수요와 품질 유지가 점점 어려워지고 있는 상황이다.

일상 생활에서 이동 단말 시스템의 쓰임새가 점차 넓어지면서, 이에 필요한 보다 큰 규모와 복잡도를 갖는 임베디드 소프트웨어를 개발해야 한다. 하지만, 임베디드 소프트웨어의 경우는 동일한 적용 분야에서도 기반하는 하드웨어 플랫폼과 응용 소프트웨어의 요구사항이 매우 상이하여 기존의 전통적인 소프트웨어 개발 방식으로는 품질과 납기일을 맞추기가 어려운 상황이다. 기술적 혁신 없이 일력을 많이 투입하여 개발 기간을 단축시키는 것은 소프트웨어공학적 관점으로 볼 때 회의적이며, 수요가 항상 증가하는 것은 아니기 때문에 회사 재정 문제에 있어 성장에 걸림돌이 될 수 있는 중요한 문제이다.

지금까지 이동 단말 시스템이 기반하고 있는 기술과 플랫폼들은 매우 다양하다. 이동 통신 기술로써 CDMA와 GSM이 세계적으로 널리 사용되고 있으며, 마이크로소프트 사의 Windows Mobile, Windows CE, 그리고 리눅스, Rex, Simbian 등이 이동 단말 시스템의 운영체제로 사용되고 있는 상황이다. 또한, 하드웨어 플랫폼에 장착되는 주변 장치도 다양한 조합으로 구성되기 때문에 카메라, LCD, 플래시 메모리, 무선랜, 적외선통신, 시리얼통신, 블루투스 등의 장비에 대한 다양성에 대해서도 고려해야 한다. 이렇듯, 이동 단말 시스템에 존재하는 소프트웨어 상의 이종성(heterogeneity)을 극복할 수 있어야만이 제품 생산성을 극대화 할 수 있는 것이다.

이동 단말 시스템이동단말 시스템의 이종성을 최소화하기 위한 노력으로 다양한 연구가 수행되어 왔다. 예를 들어 마이크로소프트 사의 Microsoft Radio Interface Layer(RIL)는 운영체제 상의 프로세스와 이동 통신 모뎀제어 소프트웨어 사이에 논리적 층을 구성하여 핫 스폿(hot spot) 형태로 다양한 플랫폼에 이식할 수 있도록 돕는다. 응용 어플리케이션 쪽으로는 TAPI, SMS API, SIM API 등의 전화 관련 표준 인터페이스를 이용하고 있으며 무선 모뎀 쪽으로는 AT 커맨드를 통해 통신 함으로써 채택 기술의 이종성을 극복하는 기본 구조를 제공하고 있다. 또한, 주목할 만한 연구로 Danger[1] 사의 HipTop 시스템을 들 수 있겠다. HipTop은 자바로 개발된 이동 단말 소프트웨어 이다. 따라서, 기본적으로 플랫폼 독립적인 자바의 속성에 대해 그대로 혜택을 받고 있다. 따라서, 이동 단말 소프트웨어 개발 작업이 자바 가상 기계(JVM)을 하드웨어에 이식하는 작업으로 단순화 된다. 이동 단말 소프트웨어는 이미 JVM 상에서 검증 받았기 때문에 품질 시험 시간이 획기적으로 줄어 들게 되는 것은 가장 큰 장점 중에 하나가 될 것이다. 하지만 C나 C++로 개발된 소프트웨어와 비교하면 성능이 많이 떨어지고 사용자 어플리케이션의 요구 사항이 고정되어 있어 이를 확장 및 수정하려면 역시 비용이 많이 든다는 점이 단점으로 작용한다. 또한, J2ME의 미들렛(Midlet)의 경우 최초에는 단순한 게임 등의 용도로 시작되었으나 최근에는 이동 통신 사업자의 다양한 서비스를 위한 어플리케이션 성격으로 많이 발전되어 있는 상태이다. 이런 경향을 볼 때, 머지 않아 이동 단말 소프트웨어가 모두 자바로 구현되어 있어서 사용자의 소프트웨어 개발자는 오직 사용자가 필요로 하는 기능적 요구사항의 이종성에 대해서만 고려하면 되는 상황이 올 것으로 예상된다.

본 논문은 임베디드 소프트웨어, 특히 이동 단말 사용자 소프트웨어 개발 시에 복잡한 기능적 요구사항 조합에 대해 얼마나 효과적으로 소프트웨어공학 적인 방법을 이용하여 처리할 수 있는 지에 초점을 맞추고 있다. 기존의 객체지향이나 컴포넌트 기반 개발 기술 이용 만으로는 얻을 수 없는 도메인-특정 추상화(domain-specific abstraction) 기법을 이용함으로써 보다 높은 수준의 추상화를 구현할 수 있는 것이다. 이동 단말 소프트웨어는 모뎀 제어를 위한 소프트웨어와 사용자에게 응용성을 제공하기 위한 사용자 소프트웨어로 크게 분류할 수 있는데 사용자 소프트웨어의 경우 사용자 인터페이스를 구현하기 위한 부분과 모뎀 제어 소프트웨어와 통신을 담당하는 부분으로 나눌 수 있겠다. 이동 단말 사용자 소프트웨어 개발에는 상당히 많은 인력을 필요로 하며 그 중에서도 특히 사용자 인터페이스 관련하여 작업량이 많다. 비슷한 제품을 시리즈로 개발하는 경우 이전 개발 산출물에 상당부분 이미 구현이 되어 있는 경우가 많지만, 다른 그래픽 디자인 또는 다른 사용자 시나리오를 제공하는 경우 기존의 설계 및 코드를 적극적으로 이용하기 어려운 것이 현실이다. 이동 단말 시스템이동단말 시스템이 제공하는 기능에 대해 사용자에게 보여지고 입력할 수 있는 장면 단위로 표현하고 이를 소프트웨어 아키텍처로 삼아 설계, 구현 단계에 반영한다면 사용자 인터페이스 구현에 대한 비용이 상당 부분 감소할 것으로 예상이 되며, 또한 기존의 개발 산출물에서 필요한 기능들을 컴포넌트화하여 재사용할 있는 기회가 크게 증대 될 것이다.

나머지 논문 구성으로 2장에서 배경 연구에 대해 설명하고, 3장에서는 본문으로 제안되는 접근 방법에 대하여 상세하게 설명한다. 4장에서는 적용 사례를 통해 그 유용성을 검증해 보며 5장에서는 결론과 향후 과제에 대해 언급할 것이다.

2. 배경 연구

소프트웨어공학 연구 분야에서 가장 중요한 목적 중에 하나는 개발 및 유지보수 과정에서 높은 수준의 추상화 수준을 제공하는 것이 될 것이다. 높은 추상화 수준은 개발 과정에서 발생 할 수 있는 우발적 혹은 비본질적 복잡도(accidental complexity)[1]를 감소시키고 소프트웨어의 모듈화를 향상시키는 역할을 하게 된다. 현재까지 다양한 수준의 소프트웨어공학적 추상화 기법을 제안하는 연구가 많이 존재하였으며 현재 많이 사용되는 객체지향 기법, 컴포넌트 기반 개발 기법, 프로덕트라인, MDA(Model-Driven Archit-

1) http://www.danger.com 참조

ecture) 기법도 맥락을 같이 한다 할 수 있겠다. 본 장에서는 객체지향 연구가 제안된 이후에 이를 보완하기 위해 수행되어 온 대표 연구에 대하여 살펴보고 이러한 연구가 결과적으로 어떻게 프로덕트라인공학을 구현하기 위한 기반 기술 역할을 하고 있는지 살펴 본다.

사용자 요구사항은 유즈케이스(usecase) 형태로 기술할 수 있다. 소프트웨어개발 생명주기의 분석 단계를 통하여 유즈케이스 모듈과 상호 관계성이 파악되며 이들은 점차 구조화되고 높은 모듈화 수준을 갖게 된다. 이 단계에서 다뤄지는 분석 모델(analysis model)은 철저히 사용자 관점에서의 소프트웨어 기능에 대한 구조와 모듈성을 갖게 개발 된다. 따라서, 유즈케이스 모델 수준에서 사용자 위주의 변경 요구를 반영하면 변경 영향 범위를 최소화 할 수 있는 장점을 얻을 수 있는데, 불행히도 유즈케이스 수준의 모듈화는 설계, 구현 단계를 거치면서 상당 부분 그 실체를 잃게 된다. 이처럼 사용자 수준에서 인지하는 기능적인 모듈성과 설계, 구현을 위한 개발자 관점에서의 모듈화는 목적에 따라 차이점을 갖게 되는데, 이 두 가지 모듈화 개념을 일치시키기 위해서는 소프트웨어공학적 기술이 필요하다. 주요 원인으로는 크게 두 가지를 생각 할 수 있겠다. 첫째, 소프트웨어에 존재하는 비기능성(예: 시스템 성능, 메모리사용 최소화, 고장허용성, 데이터지속성) 등이 기능성 관점의 모듈화 특성을 유지하기 어렵게 만든다. 둘째, 모듈 내부에 정의된 기능에 대해서는 강한 모듈화가 제공되지만 모듈 사이에 관계된 기능성에 대해서는 시스템 구성요소로서의 모듈화 지원이 부족한 상황이다.

협업기반설계(collaboration-based design) 기법은 객체지향 기법에서의 주요 모듈인 클래스에 대하여 클래스들 사이에 존재하는 기능적 행위에 대해 관심사의 분리(separation of concerns)를 지원하기 위한 연구로 제안 되었다. 여러 종류의 클래스가 참여하는 협업 행위는 본질적 특성상, 가정하고 있는 응용 상황(application context)에 적지 않은 의존성을 갖게 되지만 한 개의 클래스만으로 구현되는 단일행위는 클래스 구조에 따라 높은 모듈화 수준을 갖게 되며 응용 상황에 크게 의존하지 않는 특성이 있다. 객체지향 프로그래밍 언어가 제안될 때만 해도 협업행위에 대해 모듈성을 제공하는 것이 고려되지 않았기 때문에 현대 소프트웨어 공학의 기술적 요구 수준으로 바라볼 때 모듈화 수준에 한계가 있는 것이다. 이렇듯, 협업행위에 대한 관심사의 분리가 지원되지 않으면 기존 단일 클래스 수준의 재사용성도 안 좋은 영향을 받을 뿐만 아니라 현재 많이 개발 되는 유형인 임베디드 소프트웨어와 같은 경우 상당 부분이 협업행위라는 점을 생각해보면 전체 기능적 측면의 재사용성에도 한계를 갖게 되는 것이다. 이를 해결하기 위한 대표 연구로 협업기반설계 방법을 제안하는 연구가 객체지향 기술 출범 이후 최근 10년 동안 많이 제안되었다. 이 중에 컨트랙트(Contracts)[2], 역할모델(roles and role model)[2, 3, 4, 5], 적응적 프로그래밍(adaptive programming)[6], 주체지향 프로그래밍(Subject-Oriented Programming)[7], 믹스인(Mixin layers)[8], 젠보카(GenVoca)[9] 등의 연구가 대표적인 연구라 할 수 있겠다.

객체지향 프로그래밍 기법이 항상 최고 수준의 기술적 지원을 하진 않음을 주장하는는다는 객체지향 기술의 한계를 지적하는 연구가 많이 있었다. 협업기반설계도 이런 연구에 속하지만 객체지향 기술 제안 시기부터 한계로써 지적되어 온 부분이 바로 시스템 성능에 대한 부분이었다. 대규모의 엔터프라이즈 급 비즈니스 소프트웨어의 경우 주로 기능성만으로 복잡성을 지니고 있기 때문에 객체지향 기법이 아주 잘 맞았지만 작고 복잡한 모듈의 실시간성 이벤트 교환을 통한 임베디드 소프트웨어의 경우 객체지향 기술의 적용 범위 밖이라는 의견들이 대세를 이루는 경우가 많았다. 객체지향 프로그래밍 언어가 기존의 언어에 비해 동적 결합(dynamic binding) 요소를 많이 갖고 있기도 하지만 사실은 시스템 성능과 같은 비기능적 요소가 소프트웨어의 기능성에 초점을 둔 모듈화와 협정관계(trade off)에 있기 때문이다. 따라서, 둘 중에 한 요소는 소프트웨어 개발 시에 포기해야 하는 경우가 대부분이었다. 이러한 문제에 대해 적극적인 해결책으로 제시된 것이 바로 관점지향 프로그래밍 기법(aspect-oriented programming)[10] 이다. 전통적인 프로그램 모듈에 의해 모듈성을 보장 받지 못하는 관심사(cross-cutting concerns)에 대해 관심사의 분리를 지원하자는 것이 관점지향 프로그램의 핵심 논리이다. 현재는 AspectJ가 대표적인 언어로서 인정받고 있다.

어떤 분야의 공학적 기술 수준이 생산성과 품질 측면에서 성공을 거두기 위해서는 대개 컴포넌트 기반의 개발 형식(component-based development)을 지원하게 되는데, 현대 소프트웨어공학의 경우도 이러한 추세를 반영하고 있다. 컴포넌트 기반 개발 방식을 사용하기 위해서는 기존의 객체지향 개발 기법을 가지고는 많은 한계를 갖고 있는데 소프트웨어 아키텍처 함께 블랙박스 결합성(black-box composition)을 지원하기 위한 기술적 지원이 현재 컴포넌트 기반 개발 기법의 기본 골자가 된다. 소프트웨어 아키텍처는 소프트웨어 시스템을 구성하는 전체적인 구조(gross structure) 와 시스템 전반에 대한 논리를 표현 제어 흐름(global control flow)에 대하여 추상화 기법을 제공해야 한다[11, 12]. 하지만, 현재까지 제안된 소프트웨어 아키텍처들 중에는 파이프 앤 필터(pipe-and-filter), 코바 컴포넌트 모델(CORBA component model), C2 아키텍쳐[13] 스타일 등의 단순한 이벤트 통신 형태에 대해서만 성공 사례가 있으며 일반적인 어플리케이션에 대한 모듈 결합 프로그래밍(module interconnection language)[14] 관점에 초점을 둔 개발을 지원할 수 있는 대표적인 소프트웨어 아키텍처를 위한 기술적 수준은 아직 많이 부족한 상태이다.

컴포넌트 기반 개발 기술의 다음 단계로서 보다 높은 생산성과 품질을 제공할 수 있는 것이 바로 프로덕트라인 공학이다. 이는 구체적인 제품의 하나 하나에 대하여 개발 주기 전체를 모두 거쳐야 하는 일반 개발 기법과는 차별되게 뚜렷하게 공통성을 갖고 있으나 쓰임새 의 변이성(vari-

ation)을 많이 갖고 있는 제품 군(product family)에 대해 개발을 진행 완료하여 후에 계속되는 같은 도메인의 제품 개발 대해 높은 생산성과 품질을 보장 할 수 있는 기술이다. 소프트웨어공학에서는 유사한 도메인을 갖는 소프트웨어에 대해 소프트웨어개발주기 전반에 대한 산출물을 기반을 하여 구체적인 특성을 갖는 소프트웨어를 적은 비용으로 개발할 수 있도록 지원한다. 현재 표준적인 객체지향 설계 언어인 UML과 함께 이러한 프로덕트라인 개발 방식을 지원하고자 하는 연구로 MDA(Model-Driven Architecture)이 있겠다[15]. MDA의 단어적인 의미로 볼 때 단순히 설계 차원에서 개발이 완료될 수 있음을 암시하지만 실재 기술적 구조를 보면 플랫폼에 독립적인 모델(PIM) 과 플랫폼에 종속적인 모델(PSM)을 분리하여 정의하여 하나의 PIM으로부터 다양한 플랫폼에 대한 PSM을 변역 기법을 이용하여 다양하게 만들어 낼 수 있는 기술을 의미한다. 이러한 측면에서 MDA는 프로덕트라인 공학에 대한 플랫폼에 대한 변이성에 대한 해법을 제시하고 있다고 할 수 있겠다. 이와 함께, 피처지향 프로그래밍(feature-oriented programming)은 소프트웨어의 기본 단위를 피처로 정의하여 개발, 확장, 변경 할 수 있는 기법을 말한다. 피처 수준의 변경은 구현 단계의 프로그램 코드에까지 영향을 주도록 프로그램 합성 기법을 이용한다. 대부분의 경우 피처지향 프로그래밍은 피처에 대한 적절한 표현 방법으로 도메인 기술 언어(domain-specific language)를 이용하는데, AHEAD와 ATS가 피처지향 프로그래밍을 구현하고 있는 대표적인 이론과 도구로써 제안되었다[16, 17]. AHEAD는 젠보카의 후속 연구로 수학의 대수식(algebraic expression)을 이용하여 프로그램을 단계 별 개발 및 확장 할 수 있는 방법을 제공한다. 이런 의미에서 피처지향 프로그래밍 기법도 광의의 프로덕트라인 공학 기술의 한 연구분야로 볼 수 있겠다.

소프트웨어 프로덕트라인 공학 기법은 미국방부(U.S. Department Of Defense) 의뢰의 소프트웨어 개발 산업에서는 그 성공 사례가 가끔 보고 되었지만 일반 산업체에서는 성공 사례를 많이 찾아 보기 어렵다. 이 중에 일반 산업체로서는 필립스가 가전 제품 군(TV, DVD, 레코더 등)에 대하여 프로덕트라인 기법을 대한 주목할 만한 연구를 수행하여 그 결과를 학계에 보고한 대표적인 사례라 할 수 있겠다[18]. Koala라는 아키텍처 기술 방법을 이용하여 시스템을 기술하고 임베디드 소프트웨어 형태로 적합한 컴포넌트 플랫폼을 제안하여 필립스 가전 제품 군 중에 TV, DVD, 레코더 등에 공통적으로 사용되는 기능들을 컴포넌트화 하여 TV 등과 같이 한 가지 독립된 도메인보다 몇 가지 기능적 공통성을 지닌 도메인 집합에 대하여 이른바 프로덕트 파퓰레이션(product population) 기법을 제안 하였다. 하지만, Koala는 단순히 스위치 구조를 갖는 컴포넌트 커넥터를 통하여 피처의 변이성을 다루고 있으며 도메인에 관한 높은 추상화 수준의 피처 결합 규칙을 제공하고 있진 않다. 특정 도메인에 대한 결합 규칙 없이는, 재사용성 측면에서 소프트웨어의 생산성은 강한 한계를 안고 있으며 이는 젠보카 연구에 대한 한계라고 할 수 있겠다. 도메인 기술 언어는 특정 도메인에 대하여 충분한 표현력과 밀도 있는 언어 구조를 지니고 있어야 한다. 대표적인 성공을 거둔 도메인 기술 언어로는 데이터베이스 도메인을 위한 SQL과 이미지 처리 시스템 도메인을 위한 파이프 앤 필터 형식 기술 언어가 있다.

현대의 전형적인 임베디드 시스템을 보면, 이동 단말 시스템이동단말 시스템의 경우 가장 많은 기능상의 변이성을 갖고 있으며 또한 다양한 제조사에서 수년간의 후속 모델로서 제품을 꾸준히 생산하고 있기 때문에 소프트웨어 프로덕트라인 기법을 적용하기 가장 적합한 도메인이 될 수 있을 것이다. 노키아와 모토롤라 등의 제조사가 이에 관심을 갖고 투자하고 있지만 아직 내세울 만한 프로덕트라인 공학이라고 부를 만한 연구 성과가 부재한 상태이다. 본 논문에서는 이동 단말 소프트웨어에서 구현하고 있는 피처 단위의 기능성 들을 사용자 인터페이스 관점에서 다양한 시나리오로 구성할 수 있는 도메인 기술 언어를 제안하고 이를 바탕으로 소프트웨어 아키텍처를 형성하여 사용자 요구 수준의 모듈성을 구현 단계까지 유지할 수 있는 방법을 제안한다. 이에 앞서 저자가 3년 동안 수행한 PDA 스마트폰 양산 프로젝트에 대한 간단한 수개와 이를 통해 얻은 경험을 정리하여 프로덕트라인 공학의 필요성에 대해 강조하려 한다.

3. MDA 기반 이동 단말 소프트웨어 개발 방법

3.1 기본 개념

도메인 기술언어는 개발자로 하여금 특정 도메인에 속한 어플리케이션 구성에 필요한 도메인 어휘(vocabulary)를 이용해 높은 추상화 수준에서 소프트웨어를 개발할 수 있도록 지원 한다. 도메인 기술언어는 설계 및 구현 과정에서 발생하는 비본질적 복잡성에 대하여 문제 해결 방법을 가능한 한 개발자에게 지우지 않게 하여 보다 창의적이고 근본적인 업무에 집중 할 수 있도록 한다. 도메인 기술 언어를 통해 작성된 모델이 문법적으로 맞기만 하면 이미 검증되고 테스트된 구체적인 플랫폼의 설계, 구현 모델로 자동 변경되는 것이다. 소프트웨어 플랫폼은 MDA 문서에 나오는 것처럼 다음과 같이 정의가 될 수 있겠다. 소프트웨어 플랫폼은 구현에 필요한 구체적인 세부사항들을 숨기면서도 이를 지원하는 상호 연관성 높은 기능성의의 집합을 제공하는 기술과 서브시스템의 집합을 의미한다[15].

MDA의 주요 특징은, 보다 구체화되는 모델로의 전환을 위해, PIM과 CIM으로부터 PSM으로의 자동화된 변환을 요구하고 있다는 것이다. 개발 과정에서 지원되는 추상화 개념의 수준으로 볼 때, 일반적인 설계언어를 통한 개발과 도메인 기술 언어를 통한 개발은 분명한 차이가 있다. 본 장에서는 이동 전화 단말기 소프트웨어 개발을 위한 도메인 기술 언어(Menu-Navigation Viewpoint DSL)를 제안한다. 사용자가 이동 단말 시스템이동단말 시스템을 사용할 때 메뉴를 통한 필요한 기능을 제공하는 화면으로 이동하는 데

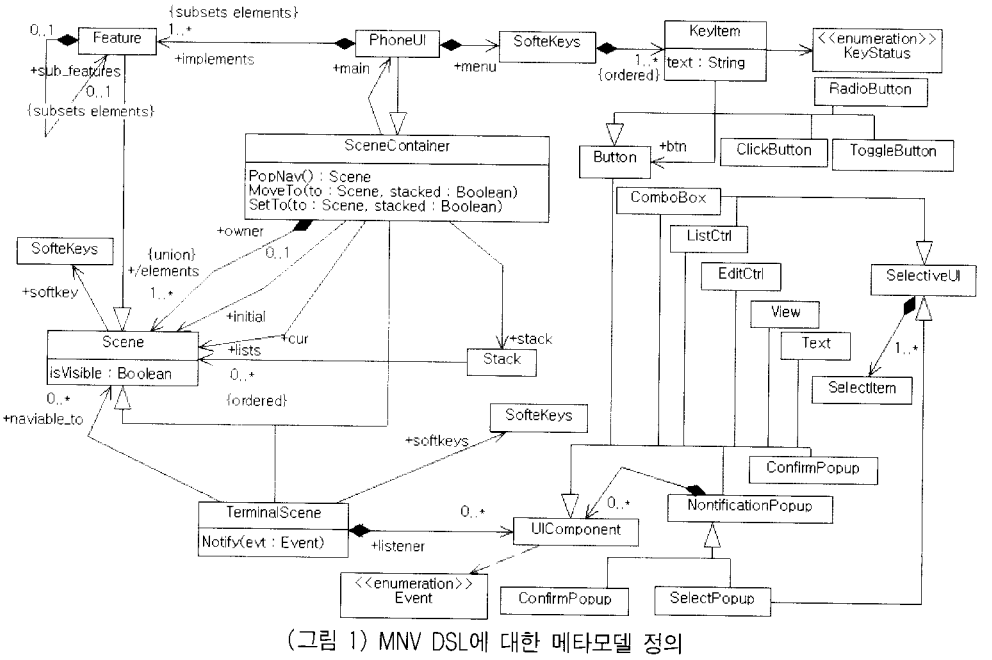
착안하여 개발 되었다. 또한, 이동 단말 시스템이동단말 시스템의 사용자 수준 기능을 설계하는 엔지니어가 시스템을 개발 할 때 사용자에게 보여지는 메뉴와 장면 위주로 기술되는 점을 볼 때, 이동 단말 시스템이동단말 시스템의 사용자 소프트웨어 도메인에 대해 MNV DSL이 충분한 표현력을 갖고 있음을 알 수 있다. 개발자가 같은 관점으로 시스템을 기술함에 따라 사용자 관점의 기능성을 개발 단계에서도 계속 유지할 수 있게 되어 유지 보수 단계에서도 높은 모듈화 수준을 제공하게 된다. 이렇게 기술된 시스템 모델은 특정 GUI 플랫폼에 대하여 정의된 변환 규칙에 따라 구현 단계의 모델로 전환되어 소프트웨어 아키텍쳐로써 사용된다. 검증을 위해 파일럿 프로젝트 한 형태로 Windows Mobile 5.0 GUI 플랫폼에 대하여 MNV DSL 변환 규칙을 정의하여 적용해 보았다.

3.2 도메인 기술 언어 정의: MNV DSL

제안된 도메인 기술언어: *Menu-Navigation View-point*(MNV)는 이동 단말 시스템이동단말 시스템의 사용자 소프트웨어를 기술하기 위한 목적으로 정의되었다. 대부분의 이동 단말 시스템이동단말 시스템의 사용자 소프트웨어는 어떤 전형적인 구조와 행위를 갖고 있는데, 이 중에서도 사용자에게 각 서비스를 제공하는 피처 단위의 기능들에 대해 메뉴를 통한 장면 단위의 접근 방식을 사용하는 큰 공통점이 있다. 또한, 각 장면 단위는 스택 구조의 프로토콜을 통해 이벤트나 사용자 입력 등의 인터럽트에 대해 발생한 장면에 대해 이전 장면으로의 안전한 복귀가 진행되도록 하고 있다. 그리고, 소프트키 개념을 제공하여 사용자의 입력을 정규화 하고 있다. 제조사에 따라 2개 혹은 3개의 소프트키 개념을 제공하고 있는데, 사용자의 직관적 명령 형태에 따라 장면 단위의 명령을 특정 위치의 소프트키에 적절히 할당하고 있다. 이러한 공통성에 기반하여 NNV DSL을 정의 한다.

전체 시스템 관점에서 보면, 이동 단말 서비스라는 하나의 피처에 대하여 장면을 전환 시키면서 사용자에게 서비스를 제공하게 된다. 또한, 각 장면들은 상태에 맞는 메뉴들 제공하여 사용자에게 적절한 판단을 하여 제공되는 기능들을 필요에 따라 조합하여 사용 할 수 있도록 하는데 이러한 과정에서 기존에 보여졌던 장면들이 데이터를 유지한 체 다시 보여져야 할 수도 있으며 음성호, 메시지, 주소록, 데이터 통신, 카메라 및 멀티미디어 기능, 웹 브라우징 등의 피처들에 대해 복합 사용 시나리오를 제공해야 하는 경우도 있다. (그림 1)에서는 위에 기술한 도메인에 대하여 메타모델로 정의하고 있다.

제시된 메타모델을 보면, 전체 시스템(*PhoneUI*)과 피처(*Feature*) 관계를 장면의 연속 관계로 표현하고 있다. 또한 *PhoneUI*와 *Feature*는 장면(*Scene*) 클래스의 상속 구조로 연관 지어 정의되어 있음을 알 수 있는데, 각 장면(*Scene*)들은 사용자가 제어 가능한 소프트키를 갖고 있으며 다시 하부 장면구조를 가질 수 있는 장면 컨테이너(*Scene Container*)가 될 수 있도록 구조적으로 정의되어 있다. 장면 컨테이너는 하부 장면들을 스택 프로토콜을 이용하여 관리할 수 있는 기능을 담고 있다. 장면 컨테이너 클래스의 연산자를 보면 하부 장면 전환을 위하여 *SetTo()*, *MoveTo()*, *PopNav()*를 정의하고 있다. 연산자 *MoveTo()*는 최근에 이동하는 장면의 최근 방문했던 상태를 유지하고 장면을 전환하라는 것을 의미한다. 이에 반에, 연산자 *SetTo()*는 전환되는 장면에 대하여 초기화 상태를 갖고 진행되도록 한다. 연산자 파라메터로 사용되는 *stacked* 옵션은 장면 전환 시에 이전 장면의 마지막 상태에 대해 스택에 넣은 후에 장면을 전환 할 것인지를 결정한다. MNV DSL의 주요한 부분들에



(그림 1) MNV DSL에 대한 메타모델 정의

〈표 1〉 MNV DSL 메타모델에 대한 OCL 정의

```
context PhoneUI inv:
TerminalScene.allInstances→select(elm:TerminalScene | elm.isVisible→size() = 1 and
SceneContainer.allInstances→forAll( eScene : SceneContainer | eScene.isVisible implies
eScene.elements.allInstances()→select(elm:Scene | elm.isVisible) = 1 ) and
SceneContainer.allInstances()→forAll( eScene : SceneContainer |
eScene.softkey = eScene.cur.softkey)

context SceneContainer::SetTo(to:Scene,stacked:Boolean)
pre: self.elements→includes(to) and stacked and self.cur.isVisible and
to.isVisible = false
post: self.elements.stack.lists→append(self.cur@pre) and self.cur@post = to and
self.cur.isVisible@post and self.cur.isVisible@pre = false and
self.main.menu@post = self.cur.softkey@pre and
self.cur@post→oclInState(Scene::Init) and
if self.elements.intersection(self.elements.stack.lists)→notEmpty() then
self.elements.stack.lists@post = self.elements.stack.lists→excluding(to) endif

context SceneContainer::MoveTo(to:Scene,stacked:Boolean)
pre: self.elements$\rightarrow$includes(to) and stacked and self.cur.isVisible and
to.isVisible = false
post: self.elements.stack.lists→append(self.cur@pre) and self.cur@post = to and
self.cur.isVisible@post and self.cur.isVisible@pre = false and
self.main.menu@post = self.cur.softkey@pre and
if self.elements.intersection(self.elements.stack.lists→notEmpty() then
self.elements.stack.lists@post = self.elements.stack.lists→excluding(to) endif

context SceneContainer::PopNav() : Scene
pre:
self.elements→includes(self.stack.lists@pre→last())
post:
let topItem = self.stack.lists@pre→last() in
if self.stack.lists→size() > 0 then
result = topItem and self.stack.lists@post = self.stack.lists@pre→excluding(topItem) and
self.cur@post = topItem and self.cur.isVisible@post and self.cur.isVisible@pre = false and
self.main.menu@post = self.cur.softkey@pre else result = null endif
```

대해서는(그림 1)와 <표 1>에서 자세하게 설명하고 있다. 장면 클래스의 속성인 *isVisible*은 터미널 장면(*Terminal Scene*)으로서 화면에 보이는 상태임을 의미하거나, 또는 장면 컨테이너로서 하부 장면 집합 중에 하나가 화면에 보이는 상태임을 의미한다. 시스템 동작 중에는 항상 전체 PhoneUI 내부에서 단 한 개의 장면에 대해 *isVisible* 속성이 참인 값을 갖도록 정의되어 있다. 터미널 장면은 실제 화면에 보이게 되는 장면을 의미하며 사용자가 인터페이스로 사용하는 제어용 UI 컴포넌트(*UIComponent*)로 구성될 수 있는데 이벤트 통신을 이용해 사용자와 의사 소통을 하게 된다.

앞에서 언급했던 것처럼, GUI 기반의 소프트웨어 시스템의 행위는 사용자에게 제공되는 서비스 관점에서 장면의 연속으로 추상화 되어 표현할 수 있다. 소프트웨어 아키텍쳐의 특정 뷰포인트(Viewpoint)는 설계와 구현 단계의 모델에 대해 그 구조를 유지하도록 해야 하는데, 제안된 MNV DSL의 경우 사용자 관점의 장면 연속의 뷰포인트로 소프트웨어개발 생명주기 전반에서 기본 아키텍쳐 역할을 하게 된다. MNV DSL에서 사용되는 다이어그램 형식의 구성 요소

Node type	Notation	Description
terminal scene		The same semantics as TerminalScene in the meta model.
scene container		The same semantics as SceneContainer in the meta model.
initial scene		The same semantics as a scene navigable via *initial* association end in the meta model.

Edge type	Notation	Description
stacked_set	——→	The same semantics as *SceneSet()* in *SceneContainer* in the meta model (*stacked*=true).
set	·····→	The same semantics as *SceneSet()* in *SceneContainer* in the meta model (*stacked*=false).
stacked_move	——●	The same semantics as *SceneMove()* in *SceneContainer* in the meta model (*stacked*=true).
move	·····●	The same semantics as *SceneMove()* in *SceneContainer* in the meta model (*stacked*=false).
terminal_stacked_set	▌·····→	A special type of edges that have the restricted source nodes, not all aggregated nodes but terminal scene children.

(그림 2) MNV DSL의 주요 표기법

들은 (그림 2)에 제시되어 자세하게 설명되어 있다.

(그림 2)에서는 MNV DSL 다이어그램의 주요 구성 요소들에 대해 노드(node) 와 연결선(edge)으로 나누어 설명하고 있는데, 노드에 대해서는 이미 메타모델과 OCL 정의를

통해 충분히 설명하고 있지만 연결선에 대해서는 부가 설명이 필요하다. 연결선은 크게 출발노드, 몸체, 도착노드 세 부분으로 구성된다. 하나의 장면 관점에서 보면 출발노드는 터미널 장면 혹은 장면 컨테이너가 될 수도 있다. 장면 컨테이너에 연결선이 출발할 경우 해당 컨테이너 안에 정의되어 있는 모든 하부 장면들로부터 시작되어 같은 도착노드로 연결되는 같은 의미성을 갖는 연결선에 대해 함축하여 하나로 표현하게 될 수 있다. 터미널 장면이 실제로 사용자에게 보여지는 장면 단위임을 생각할 때, 장면 컨테이너 하부 장면 집합 중에서 터미널 장면 집합에 대해서만 함축된 연결선 허용을 의미하기 위하여 연결선의 몸체 파트에 채워진 사각형으로 표기되는 *Terminal* 의미를 넣어 모델링 할 수 있다. 아래는 연결선에 포함될 수 있는 표현 식에 대한 문법을 정의한 것이다.

```
<edge_exp> := [ `[' <guard_exp> `]' ] [ <event_exp> ]
<guard_exp> := [ `!' ] <guard_cond> { <gcond_connective>
<guard_exp> }
<guard_cond> := `inState(' SCENE_ID `)'
<cond_connective> := `&' | `|'
<event_exp> :=  [ `!' ] <event> { <event_connective>
<event_exp> }
<event_connective> := `&' | `|'
<event> := `event(' EVENT_ID `)' | `timer(' INTEGER `)'
          | `key(' HWKEY_ID `)' | `softkey('
          SOFTKEY_ID `)' | `select(' INTEGER `)'
```

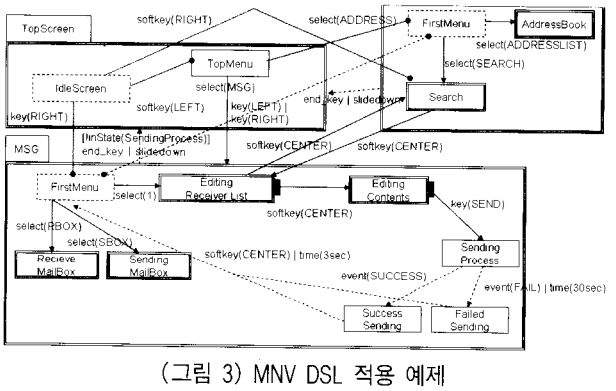
위의 문법대로 기술된 식을 이용하여 연결선이 장면 전환 과정으로 작동하기 위한 조건을 기술하게 된다. 시스템의 기본적인 이벤트 발생과 장면 상태에 대해 술어논리(predicate logic)를 이용하여 발생 조건을 표현한다. 정의에 사용된 논리 술어 *inState(sid:SCENE_ID)*은 시스템의 현재 장면이 *sid*로 표현되는 장면 노드에 도달해 잇는지 여부를 판단하기 위한 용도이다. 사용자 이벤트에 대한 자세한 정의는 <표 2>에서 설명하고 있다. 단말 시스템에 따라 슬라이드나 폴더 개폐 상태, 키 잠금 상태, 헤드 셋 장착 상태 등도 논리 술어로 등록하여 사용할 수 있다. 보다 자세한 MNV DSL에 대한 정의와 구체적인 적용에 대한 결과는 추후다른 다른 논문에서 소개할 예정이다추가적으로 확인 가능할 것이다.

〈표 2〉 MNV DSL의 이벤트

event(e:Event)	이벤트 발생을 의미하며, 집합 *Event*는 사용자 정의 이벤트로 구성된다.
timer(i:Integer)	시간 타이머 이벤트 발생을 의미함(초 단위).
key(k:HWKey)	하드웨어 키 입력을 의미하며 특정 제품에 맞도록 집합 *HWKey*는 변경될 수 있다.
softkey(s:Softkey)	소프트키 입력을 의미하며, 집합 *SoftKey*는 {LEFT, CENTER, RIGHT}나 {LEFT, RIGHT} 두 가지 전형적인 구성을 가질 수 있다.
select(id:SelectItem)	사용자가 UI 컴포넌트를 통해 선택하는 이벤트를 의미한다.

4. 적용 사례예제

MNV DSL의 유용성과 표현력에 대한 검증을 위해 이동 단말 사용자 소프트웨어에 대해 간단한 요구사항을 갖고 기술하고 이에 대하여 구현 모델로 전환하여 행위를 검증해 보았다.(그림 3)를 보면 이동 단말 시스템이동단말 시스템의 전형적인 전체 피처 중에서 대기화면, 메시지, 주소록의 피처만을 이용하여 시스템의 간단한 기능을 표현해 보았다. 시스템의 시작은 대기화면의 *IdleScreen* 장면으로부터 시작되며 왼쪽 소프트키를 이용하여 제공하는 모든 피처의 첫 번째 메뉴로 진입 가능한 최상위 메뉴 장면(*TopMenu*)으로 진입 가능하다. 현재 제공하고 있는 피처는 주소록과 메시지로 한정이 되어 있기 때문에 최상위 메뉴에서 사용자의 GUI 컴포넌트 선택을 의미하는 select() 이벤트에 의해 진입 가능하도록 기술되어 있다. 각 피처의 상위 메뉴 상에서는 오른쪽 방향키를 이용하여 순환적으로 이동 가능하다. 주소록 피처는 주소록 자체를 보여 주는 시나리오와 검색을 위한 장면을 제공하는 시나리오가 있다. 메시지 피처는 메시지 수신자를 입력하고 메시지 내용을 입력한 뒤에 발송하여 과정과 결과의 상태를 장면으로 보여주며 그 결과를 발신 메시지 보관함 장면을 통해 확인 가능하다. 메시지 피처는 수신자를 입력하는 장면에서 주소록의 검색 장면과 연결 시킴으로써 주소록 내에 등록되어 있는 사용자들을 찾아 추가할 수 있게 되는 피처 간의 상호 조합적 시나리오를 기술할 수가 있는 것이다. 본 시스템 기술에 대해 Windows Mobile 플랫폼으로 변환하여 그 행위를 검증해 보았다.



(그림 3) MNV DSL 적용 예제

5. 결론 및 향후 과제

본 논문에서는 이동 단말 시스템이동단말 시스템의 사용자 소프트웨어 개발 시에 프로덕트라인 공학 기술을 이용하여 높은 생산성과 품질을 유지할 수 있는 방법을 제안하였다. 또한, 사용자 요구 수준에서의 기능적 모듈화 구조가 설계, 구현 단계의 모듈화 구조와 같은 관점에서 기술하고 있기 때문에 사용자 수준의 변경 요구에 대하여 시스템에 미치는 영향을 최소화 할 수 있게 된다. MDA의 기본 접근 방법을 이용하여 소프트웨어 기능을 장면의 연속으로 기술

하기 위한 DSL을 이용하여 플랫폼에 독립적인 모델(PIM)을 작성하고 구체적 플랫폼 구조의 모델(PSM)으로 변환한다. 변환된 설계와 코드는 다른 세부 기능을 구현하기 위한 기본 틀 역할, 즉 소프트웨어 아키텍처의 핵심 뷰포인트로 사용될 수 있다

현재까지는 파일럿 프로젝트 수준의 MNV DSL 정의와 Windows Mobile 플랫폼으로의 변환 프로그램 정도를 개발하여 그 유용성을 검증해 본 상태이다. 향후 과제로 MNV DSL을 표준 UML 프로파일로 형태로 정의하여 UML의 기본 다이어그램 기술을 돕는 도구를 사용할 수 있도록 할 계획이다. 또한, 목적 플랫폼으로써 Windows Mobile 뿐만 아니라 Java2ME의 JWTI 프로파일을 이용한 미들렛(Midlet), 리눅스 X-windows의 Motif 플랫폼에 대해서 수학적인 모델 변환 규칙을 정의하여 제안한 방법이 보다 실용적으로 사용 될 수 있도록 그 적용 범위를 확대해 나갈 예정이다.

참 고 문 헌

[1] Frederick P. and Brooks Jr., "No silver bullet: Essence and accidents of software engineering," IEEE Computer, pp.10-19, April, 1987.

[2] Helm G, Holland I, and Gangopadhyay D, "Contracts: Specifying behvioral compositions in object-oriented systesm," ACM SIGPLAN Notices, Vol.25, No.10, pp.303-311, 1990.

[3] VanHilst M and Nokin D, "Using role components to implement collaboration-based designs," Proceedings of the ACM Conference on Objecct-Oriented Systems, Languages, and Applications, California, pp.359-369, 1996.

[4] Kristensen BB., "Roles: Conceptual abstraction theory and practical language issues", Theory and Practice of Object Systems(special issue on subjectivity in objecgt-oriented systems, Vol.2, No.3, pp.143-160, 1996.

[5] Joon-Sang Lee and Doo-Hwan Bae, "An enhanced role model for alleviating the role-binding anomaly," Software: Practice and Experience, Vol.32, No.14, pp.1317-1344, Nov. 2002.

[6] Lieberherr KJ, 'Adpaptive Object-Oriented Software Evolution: The Demeter Method with Propagation Patterns,' PWS Publishing Company, Boston, MA, 1996.

[7] Harrison W and Ossher H, "Subject-Oriented Programming(a critique of pure objects," ACM SIGPLAN Notices, Vol.28, No.10, pp.411-428, 1993.

[8] Smaragdakis Y and Dabory D, "Implementing layered designs with mixin layers," Proceedings of the European Conference for Object-Oriented Programming, Lecture Notes in Computer Science 1445, July, 1998.

[9] D. Batory and B. J. Geraci, "Composition Validation and Subjectivity in GenVoca Generators," IEEE Transactions on Software Engineering(special issue on Software Reuse), Vol.23, No.2, pp.67-82, Feb., 1997.

[10] Kiczale G and et al., "Aspect-oriented programming: Proceedings of the European Conference for Object-Oriented Programming," Finland, Lecture Notes in Computer Science 1241, Springer, pp.220-243, 1997.

[11] D.E. Perry and A.L. Wolf, "Foundations for the study of software architecture," ACM SIGSOFT Software Engineering Notes Oct, pp.40-52, 1992.

[12] M. Shaw and D. Garlan, "Software architecture: Perspectives on an emerging discipline," Prentice Hall, 1996.

[13] R. N. Tayler and et al., "A component- and message-based architectural style for GUI software," IEEE Transactions on Software Engineering, Vol.22, No.6, pp.390-406, June, 1996.

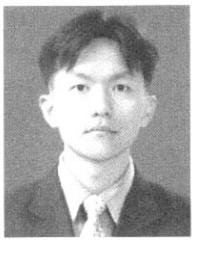
[14] F. DeRemer and H.H. Kron, "Programming-in-the-Large versus Programming-in-the-Small," IEEE Transactions Software Engineering, Vol.2 No.2, pp.80-86, June 1976.

[15] Object Management Group : MDA Guide V1.0.1. http://www.omg.org., 12th June, 2003.

[16] D. Batory and J.N. Sarvela, A. Rauschmayer, "Scaling Step-Wise Refinement," IEEE Transactions on Software Engineering, Vol.30, No.6, pp.355-370, June, 2004.

[17] J. Liu, D and Batory, S., "Nedunuri : Modeling Interactions in Feature Oriented Designs," International Conference on Feature Interactions(ICFI), June, 2005.

[18] R. van Ommering, "Building Product Populations with Software Components," Proceedings of the Twenty-fourth International Conference on Software Engineering, pp.255-265, 2002.



이 준 상

e-mail : joon@se.kaist.ac.kr

1997년 동국대학교 컴퓨터공학과(공학사)

1999년 한국과학기술원 전산학과전공(공학석사)

2003년 한국과학기술원 전산학과전공(공학박사)

2003년~현재 LG전자 디지털 미디어 연구소, HPS 그룹 선임연구원

관심분야 : 소프트웨어공학, 객체지향 기술, 소프트웨어 아키텍처 등

채 흥 석

e-mail : hschae@pusan.ac.kr

1994년 서울대학교 원자핵공학과(학사)

1996년 KAIST 전산학과(공학석사)

2000년 KAIST 전산학과(공학박사)

2000년~2003년 동양시스템즈(주) 기술연구소 선임연구원

2003년~2004년 KAIST 전산학과 초빙교수

2004년~현재 부산대학교 컴퓨터공학과 조교수

관심분야 : 객체지향 모델링, 소프트웨어 테스팅, 분산 미들웨어, 소프트웨어 아키텍처