프로덕트 라인 개발에서 피쳐 모델의 명세화 기법
(Feature Model Specification Method in Product-Line Development)

송 재 승 † 김 민 성 †† 박 수 용 †††
(Jaeseung Song) (Minsung Kim) (Sooyong Park)

요 약 빠르게 변화하는 시장의 요구에 대응하고자 특정 영역에 속하는 애플리케이션 간의 재사용을 높이려는 프로덕트 라인 개발 방법에 대한 연구가 활발하게 진행되고 있다. 프로덕트 라인 개발 방법에서는 영역 내의 여러 애플리케이션들 간의 차이점과 공통점을 분류하는데 피쳐 모델링이라는 분석 방법을 주로 사용하고 있다.

기존 피쳐 모델링에서는 피쳐를 비정형적으로 명세화하기 때문에 모호성, 이해의 오류, 잘못된 해석 등의 문제가 발생하고 있다. 피쳐를 추상화하여 도메인에 독립적인 메타 모델로 나타내고 정형화 기법을 도입하여 명세화 한다면 기존의 피쳐 모델에서 발생하는 문제점들을 해결할 수 있을 것이다. 따라서 본 논문에서는 첫째, 메타 수준에서의 피쳐 모델링을 통하여 피쳐의 구조와 속성을 정의한 후 다중 패러다임 정형화 명세 언어를 사용하여 피쳐를 명세화하는 기법을 제시하였다. 둘째, 피쳐에 대한 정형화 명세 프로세스를 기술하였으며, 셋째, 명세화된 피쳐들 간에 발생할 수 있는 문제점들을 해결하기 위한 피쳐 상호작용 관리기법을 정의하였다. 그리고 마지막으로, 제시된 피쳐의 정형화 명세 기법을 분산 미팅 스케줄러 시스템에 적용시켜보았다.

키워드 : 프로덕트 라인 개발, 정형화 명세, 피쳐 모델

Abstract In a feature modeling, problems such as ambiguities, interpretation errors, incompleteness, etc caused by informal specification occur in the modeling phase. Therefore, feature specification method and processes are suggested in this paper to resolve these problems.

The structure and language of feature modeling is defined in this paper to specify various features. First, this feature model is abstracted in the meta-level to get predicates and attributes. Formal feature model specification method is proposed using multi-paradigm language. Second, Feature specification process is proposed to describe how to specify feature formally. And third, Feature interaction management is defined to solve the problems caused between specified features. Finally, the proposed feature specification method is applied to Distributed Meeting Scheduler System domain.

Key words : Product Line Development, Formal Specification, Feature Model

1. 서 론

기존의 소프트웨어 개발에 있어서 소프트웨어 엔지니어는 각각의 소프트웨어 시스템을 개발할 때, 시스템에 따라 각 시스템에 해당하는 요구사항 분석, 아키텍쳐 설계, 구현, 문서화 작업 등을 따로 수행해 오고 있다. 하지만, 소프트웨어 시장의 규모가 점점 커지고 다양해짐에 따라 소프트웨어 시스템을 개발하는 조직들은 점차적으로 비슷한 종류의 소프트웨어를 개발하는데 있어서 매번 같은 단계를 반복함으로써 많은 비용을 소모하게 되었다. 또한, 빠르게 변화하는 시장의 요구를 충족시키고, 시장에서의 경쟁에 신속하게 대응하기 위하여 새로운 프로젝트를 개발하거나 기존의 프로젝트에 새로운 기능들을 추가 시켜야 하는 요구에 직면하게 되었다. 그러나 기존의 개발 방법으로는 이러한 문제에 대한 해결책을 제시하기 어려웠다. 따라서, 이를 해결하기 위한 새로운 개발 방법이 필요하게 되었다.

현재 많은 조직들은 소프트웨어 시스템을 개발하는데 있어 개발 프로세스의 속도를 높이기 위하여 이전 프로젝트의 프로세스를 참고하는 방법을 사용하고 있다. 이 중 소프트웨어 시스템들간의 유사성을 바탕으로 재사용성을 높이기 위하여 프로덕트 들 간의 공통적인 요소들

† 비 회 원 : LG전자 CDMA단말연구소 연구원
anthonys@lge.com
†† 비 회 원 : 서강대학교 컴퓨터학과
mskim@selab.sogang.ac.kr
††† 정 회 원 : 서강대학교 컴퓨터학과 교수
sypark@ccs.sogang.ac.kr
논문접수 : 2002년 8월 12일
심사완료 : 2003년 8월 7일

을 찾아내어 이를 컴포넌트화 하는 프로덕트 라인[1,2] 개발 방법에 대한 연구가 되어지고 있다. 프로덕트 라인 개발에서 프로젝트들은 프로덕트 패밀리(Product family)로 개발이 되어진다. 프로덕트 라인 개발 방법의 핵심은 프로덕트 패밀리들 간의 공통적인 자산(Common Asset)을 찾아내고 이를 컴포넌트화하여 재사용하는데 있다. 그러므로 프로덕트 라인의 핵심이 되는 소프트웨어 자산을 개발하기 위해서 프로덕트 라인 방법은 해당 프로덕트 라인의 공통점을 찾아내고 차이점을 관리할 수 있는 방안을 제시하여야 한다.

피쳐(feature) 중심의 영역 분석 방법인 Feature Oriented Domain Analysis method(FODA)[3,4]는 1990년 Software Engineering Institute(SEI)를 통하여 소개되었다. 현재 피쳐 중심의 분석 방법은 산업계와 학계에서 영역 분석과 프로덕트 라인을 분석하기 위한 방법으로 광범위하게 사용되고있다. 그러나 피쳐 중심의 영역 분석 방법에서 피쳐는 비 정형화된 자연어로 명세가 되고, 피쳐 모델링의 상당 부분이 영역 전문가의 경험에 의존하고 있기 때문에 많은 문제점들이 존재한다. 비 정형적 표기에서 발생할 수 있는 문제점들은 모호성, 이해의 오류, 불완전성, 그리고 피쳐 상호 작용 문제 등이 있다. 앞에서 언급되어진 문제점들은 피쳐를 정형화된 형식으로 명세화 하는 방법에 의하여 해결되어질 수 있지만, 현재 피쳐 정형화에 대한 연구가 부족한 실정이다.

목표 지향 요구공학 분야에서는 Knowledge Acquisition in Automated Specification(KAOS)[5,6]라는 방법을 사용하여 추상화 레벨에서 목표를 모델링하고 명세함으로써, 목표에 대한 증명 및 검증 그리고 충돌 등에 대한 관리를 가능하게 하였다. 따라서 목표 지향 분석 분야에서는 KAOS방법을 통하여 목표에 대한 모델링을 보다 체계적으로 실시하고, 정형화 방법을 통해 비 정형적인 명세로부터 발생하는 문제점들을 해소하였다.

그러므로 본 논문에서는 피쳐의 행동을 메타 수준에서 모델링하여 영역에 독립적인 피쳐 모델의 구조와 문법을 제시하고 이를 KAOS에서 사용하는 다중 패러다임 언어를 사용해서 정형화하였다. 또한 피쳐를 습득하고 이를 정형화 명세 하는 프로세스를 정의하였다. 피쳐 명세화 방안과 프로세스에 대한 타당성을 상호 작용 관리를 통하여 제시하고, 분산 미팅 스케줄러 시스템에 적용시켜 보았다.

본 논문의 구성은 다음과 같다. 우선, 2장에서는 관련 연구로서 프로덕트 라인 개발 방법에서 프로덕트 패밀리들 간의 공통점과 차이점을 구분하는데 사용되어지는 피쳐 모델링 기법과 정형화 기법에 대해서 설명한다. 3장에서는 기존 피쳐 모델의 문제점을 바탕으로 피쳐 메타 모델과 피쳐의 구조, 그리고 명세화 방안을 제안하고, 피쳐를 습득하고 명세화 하는 프로세스와 함께 피쳐 상호 작용 관리기법을 통해 명세화 방안의 타당성을 기술한다. 그리고 4장에서는 3장에서 제시한 피쳐의 정형화 명세 기법을 분산 미팅 스케줄러 시스템에 적용시켜 보고자 한다. 마지막으로 5장에서는 결론 및 향후 연구 과제를 기술한다.

2. 관련 연구

2.1 프로덕트 라인 개발 방법에서의 FODA

프로덕트 라인 개발 방법에서는 피쳐를 사용하여 애플리케이션간의 공통점과 차이점을 구분해 내는데 이 과정에서 FODA가 가장 많이 사용되고 있다. FODA는 영역 분석 방법 중에서 대표적인 것으로서 영역 분석과 소프트웨어 프로덕트 라인 개발을 위한 분석 방법으로 사용되어 왔으며, 영역 분석 방법들이 피쳐 중심의 분석 방법을 이용하여 영역에서의 공통점과 차이점을 분석하고 있다[7][8][9]. FODA 는 시스템의 집합 중에서 주도적인 또는 독특한 피쳐들을 식별하는 것을 바탕으로 도메인을 분석한다. 여기서 피쳐라는 것은 구현되고 테스트되고 배포, 유지되어져야 하는 기능적 추상화를 뜻하는 것으로 요구 사항이나 기능들을 의미한다. 각 피쳐들은 속성에 따라서 필수적(mandatory), 선택적(optional), 그리고 양자택일(alternative) 피쳐들로 구분되어져서 링크로써 연결되어진다. 이러한 피쳐간의 관계는 AND/OR 그래프를 통하여 나타내어진다. 그리고 모든 어플리케이션의 피쳐들이 명시되고, 정의되어지면 피쳐들은 분류되고, 구조화되어서 계층 모델을 생성하게 된다.

피쳐 분석 방법은 엘리베이터 시스템, 휴대폰 단말기 등 사용자 및 시장에 따라서 다양한 종류의 상품이 요구되고, 시장이 빠르게 변화되는 영역에 사용되고 있으며 점차 그 영역을 확대하고 있다.

피쳐는 사용자와 개발자간의 의사소통을 원활하게 해주며, 여러 애플리케이션간의 공통점과 차이점을 발견해 낼 수 있다는 장점이 있는 반면에, 새로운 시스템 또는 분석이 덜 되어진 시스템에 적용하는 데에는 부적합하며, 정형화에 대한 연구가 부족하여 현재 이에 대한 연구가 이루어지고 있다[10].

2.2 정형화 명세 기법

정형화 명세 기법은 시스템이 가져야 할 속성들의 집합들을 정형화된 언어와 추상화 된 수준으로써 표현하는 것이다. 이러한 정형화 명세기법은 현재 소프트웨어와 주변 환경이 이미 확립되어지고, 객체와 오퍼레이션들이 정확히 만들어지고 난 명세 프로세스의 후반부에서 점차적으로 많이 쓰이고 있으며, 대부분의 언어들은 이러한 하위 수준의 명세에 초점을 맞추고 있다[11].

정형화 명세 기법은 명세 패러다임에 따라 표 1과 같

표 1 정형화 명세 종류와 지원 언어

정형화 명세 기법	지원 언어
History-based	ERAE, TRIO
State-based	Z, VDM
Transaction-based	STATECHART, SCR
Algebraic	LARCH, ASL
Operational	PAISLEY, GIST

이 구분되어진다.

하지만, 이러한 명세 언어들의 제한된 사용범위, 하위 수준에서의 명세 등의 문제점으로 인하여, KAOS 같은 다중 패러다임 명세 언어 및 추상화를 지원하는 언어에 대한 연구가 이루어지고 있다.

2.3 기존 피쳐 모델의 문제점 분석 및 연구의 필요성

앞에서도 살펴보았듯이, 프로덕트 라인 개발 방법에서 피쳐 분석은 프로덕트 패밀리들 간의 공통 자산과 차이점을 구분해 내는데 중요한 역할을 한다. 하지만, 기존의 피쳐 분석 단계는 영역 전문가의 경험에 의해 습득이 되고, 비 정형적인 자연어로써 명세가 되어지기 때문에 이후 개발단계에서 문제가 발생할 수 있는 몇 가지 문제점들을 포함하고 있다.

기존의 피쳐 분석이 가지고 있는 문제점은 크게 다음의 네 가지로 분류되어질 수 있으며 이들을 그림 1의 피쳐 모델을 중심으로 설명하겠다.

'이해의 오류'는 서로 다른 의미를 가지는 두 피쳐가 비 정형적인 명세로 인하여 이해의 오류를 발생시켜 동일한 피쳐로써 이해되어질 수가 있다. 예를 들면, 미팅 참가자의 정보를 수정하기 위한 'ModifyParticipant' 피쳐와 미팅 참가자들을 수정하기 위한 'ChangeParticipant' 피쳐 사이에 이해의 오류가 발생할 수 있다.

'의미의 모호성'은 피쳐 분석가에 따라서 동일한 의미를 가지는 피쳐가 다른 표기법을 가지고 명세 되어질 수 있다. 이때 의미가 모호한 표기법을 사용한다면 동일한 의미의 피쳐임에도 불구하고, 서로 다른 피쳐로 인식되어 피쳐 모델에 동시에 존재하는 중복성 문제를 야기시킨다. 예를 들면, 분석가에 따라서 미팅을 생성하는 피쳐에 대하여 'Initiate'와 'Generate'라는 서로 다른 술어를 사용할 경우 의미의 모호성이 발생할 수 있다.

'불완전성'은 비 정형적인 피쳐의 명세로 인하여 충분한 기능이 명세에 포함되어야 함에도 불구하고 명세의 부재 혹은 모순되는 명세로 원하지 않는 기능이 포함될 수 있다. 예를 들면, 미팅이 진행되는 도중에 필요에 의해 미팅을 연장하기 위한 'ExtendMeetingDate' 피쳐에 대한 충분한 명세가 부족하여 해당 피쳐가 'ModifyMeeting' 피쳐에 포함되어질 수 있다.

'상호 작용 문제'는 피쳐들간의 통신 및 런타임 시에 예상치 못한 피쳐들간의 충돌이 발생할 수 있다. 기존에 사용되어지던 기술들은 피쳐의 비 정형적인 명세로 인하여 사용할 수 없는 문제점이 있다. 예를 들면, 미팅을 생성하기 위한 'InitiateMeeting' 피쳐가 동시에 중복되는 날짜를 요구하는 경우가 이에 해당한다.

이처럼 피쳐의 비정형 명세와 영역 전문가의 경험에 의존한 분석은 문제점들을 야기함을 알 수가 있다.

피쳐 모델이 가지고 있는 이해의 오류, 모호성, 불완전성 등의 문제점들은 소프트웨어 시스템을 개발하는

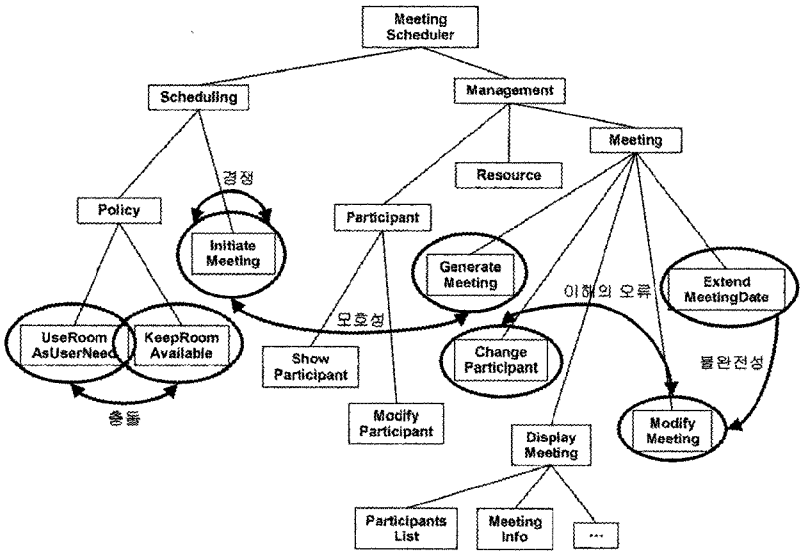

그림 1 기존 피쳐 모델의 문제점

다른 단계에 있어서도 동일하게 발생하는 문제들이다. 이러한 문제들에 대해서 기존의 다른 단계에서는 정형화 기법을 통하여 문제를 해결하기 위한 방안을 모색하였다. 즉, 수학적인 수식을 통하여 서술하고자 하는 바를 명확하게 명세함으로써, 정확한 의사소통을 가능하게 하고, 모호성을 제거하여 원하는 시스템을 만드는 것이다. 따라서, 피쳐 분석에서도 이러한 정형화 명세 기법을 적용하여 피쳐의 의미를 기술한다면, 앞에서 발생하였던 여러 가지 문제점들에 대한 해결책을 제시하고, 기존 상호 작용 관리에 적용되던 기술들을 적용할 수 있을 것이다.

3. 피쳐 모델의 정형화 명세 기법 및 검증

본 장에서는 피쳐의 행위를 메타 수준에서 모델링하여 피쳐의 구조와 언어를 UML(Unified Modeling Language)[12]을 통하여 정의하고, 이를 다중 패러다임 정형화 명세 언어를 사용하여 명세화 하는 방안과 피쳐를 정형화 명세하기 위한 피쳐 명세화 프로세스를 정의한다. 명세화 방안의 검증 단계로써 피쳐 상호 작용 관리[13]의 검증을 통해서 타당성을 제시한다.

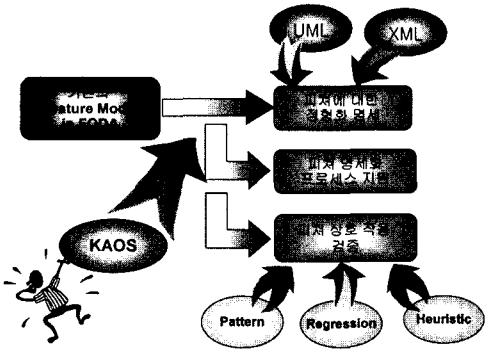

그림 2 피쳐 명세화 방안

본 논문에서는 그림 2에 나타난 바와 같이 기존의 피쳐 모델에 대하여 정형화 명세를 적용한다. 본 논문에서 제시하는 정형화 명세를 통하여 기존 피쳐 모델이 가지는 문제점들을 해결할 수 있으며, 이를 피쳐 상호 작용 관리를 통하여 검증을 한다.

3.1 적용 방법

피쳐는 그 특성상 다양한 시스템들의 여러 가지 특징들을 포함한다. 즉 시스템의 기능적인 면, 비 기능적인 면, 제한 사항 등의 모든 특징이 되는 사항들을 포함하고있다. 따라서 피쳐를 정형화 명세 하기 위해서는 다중 패러다임의 정형화 언어를 지원해야 하며, 2.3절에서 살펴본 문제점들을 해결하기 위해서는 다음과 같은 사항들을 포함하여야 한다.

첫째, 정형화 명세 패러다임을 포함한다. 피쳐는 시스템의 기능 및 비 기능적인 면 뿐 아니라 모든 시스템의 특징을 추상화시켜 표현해야 하는 특성상 다중 패러다임을 지원해야 한다. 둘째, 명세화 방안에 대한 타당성 제시를 위하여 기존 시스템 기술들을 사용할 수 있는 상호 작용 문제이다. 셋째, 명세를 위하여 피쳐를 정형화하여 나타낼 수 있는 정형화 명세 언어를 지원해야 한다. 넷째, 잘 정의된 습득 프로세스를 가지고 있어야 한다. 다섯째, 기능적인 측면뿐만 아니라 시스템의 비 기능적인 측면도 명세화 할 수 있어야 한다.

본 논문에서는 피쳐를 정형화 명세하기 위하여 필요한 요소들을 기존의 정형화 방법들과 비교를 하였다. 이 결과 KAOS에서 제공하여 주는 방법이 피쳐를 정형화 명세하기에 가장 적합한 방법임을 알 수 있었다.

표 2 정형화 명세 방법들의 비교

	KAOS	Z	VDM	UML
정형화 명세 패러다임	Multi	Single	Single	Single
상호작용문제	Yes	No	No	No
정형화명세언어	Yes	Yes	Yes	Semi
습득 프로세스	Yes	No	No	No
비 기능적 측면	Yes	No	No	Yes

표 2는 이러한 요소들을 고려하여, UML[12], Z[14,15], 그리고 VDM[16] 등의 정형화 방법들을 비교한 결과를 보여주고 있다. 여러 가지 정형화 명세 방법 중 피쳐의 명세화 방안에 적용하기 가장 적합한 방법은 다중 패러다임 언어를 제공하고, 습득 프로세스와 비 기능적인 요소에 대한 명세를 정의해 놓은 KAOS방법이 가장 적합한 방법임을 확인할 수 있다.

하지만, KAOS는 목표지향 분석 분야에서 목표를 모델링하는 과정에서 쓰이는 방법이므로, 피쳐를 모델링하고 명세하기 위해서는 해당 방법을 피쳐에 맞게 적용할 필요가 있다. 따라서 본 논문에서는 목표지향 분석에서 쓰이는 KAOS를 적용하여 피쳐의 명세에 맞게 적용하고자 한다.

표 3과 같이 제안하는 방법은 영역에 독립적인 피쳐의 구조를 만들기 위하여 메타 수준에서 피쳐를 모델링하여 구조와 문법을 정의하여 이를 UML을 통하여 표기한 뒤, KAOS에서 제시하는 명세 언어인 Z와 Temporal Logic을 사용하여 피쳐에 대한 정형화 명세를 제안하고 XML[17]로 표기하여 나타낸다. 또한, 정형화된 명세를 지원함으로써, 기존 상호작용 문제에 사용되어졌던 여러 가지 기술들, 즉 목표 회귀, 패턴 그리고 휴리스틱한 방법들을 피쳐 상호작용에도 적용할 수 있도록

표 3 KAOS와 제안하는 방법의 비교

	KAOS	제안하는 방법
목표	Goal에 대한 정형화 표기와 습득 프로세스 지원	피처에 대한 정형화 표기를 통해 모호성, 이해오류, 불완전성 등을 해결하고 상호작용 문제 지원
대상	Goal	피처
구조	Conceptual Meta-Model	메타 수준에서 피처를 모델링한 뒤 UML로 표기
언어	Z와 Temporal Logic	Z와 Temporal Logic
표기	Customized	XML을 사용하여 명세
상호작용	Systematical Techniques 지원	Systematical Techniques 지원
프로세스	정의됨	정의됨

한다

3.2 UML을 사용한 메타 수준의 피처 모델

피처는 시스템의 다양한 특징을 나타낸다. 영역에 독립적인 피처의 정형화 명세 언어와 구조를 나타내기 위해서는 다양한 피처의 행위를 추상화하여 모델링 하여야 한다. 메타 수준으로 피처들을 추상화함으로써, 피처의 구조를 영역에 독립적으로 정의할 수 있다.

이러한 메타 수준의 피처 모델링은 정형화 명세에 있어서 피처 명세화에 필요한 술어들에 대한 정의를 해준다. 정형화 명세에 필요한 술어들을 사전에 정의함으로써, 비슷한 의미의 다른 용어를 사용함으로써 발생하는 오류들을 방지할 수 있다. 또한, 습득 프로세스의 구조를 나타내며, 메타 수준의 모델링을 통하여 영역에 독립적인 피처의 구조를 제공한다.

그림 3은 영역 수준의 피처를 메타 수준으로 추상화하여 UML로 표기한 모습을 보여주고 있다.

그림 3은 'InitiateMeeting'이라는 피처를 메타 수준에서 추상화하여 UML로 명세화 한 과정을 보여주고 있다. 'InitiateMeeting' 피처는 Initiator가 새로운 미팅을 초기화하는 것을 의미하는 피처이다. 실행 조건으로는 초기화하려는 미팅이 이미 존재하지 않아야 하고, 기존 미팅과 일정이 교차되어서는 안되며, 미팅의 참가자들에게 참가 가능한 미팅이 되어야 한다는 사항들을 가진다. 미팅은 여러 자원들의 가용성을 보장하기 위하여 2주 이내에서 이루어 져야 한다는 제한 사항을 가진다. 따라서, 영역 수준에서의 피처의 행위들을 정의하는 의미들은 영역에 독립적인 피처의 구조를 위하여 추상화된다.

표 4 에서와 같이 모든 피처들은 추상화를 통하여 메타 수준으로 모델링 되어질 수 있고, 메타 수준에서의 여러 요소들은 피처를 정형화 언어를 통하여 명세화 할 때 술어로 사용되어질 수 있다.

표 4 InitiateMeeting 피처에 대한 메타 수준의 추상화

영역 수준	메타 수준
Initiator	Actor
Request	Action
Not Exist	Condition
Not overlap to others	
Available to all participants	
Duration less than 2 weeks	Constraints
Initiating	Association
Meeting	Element

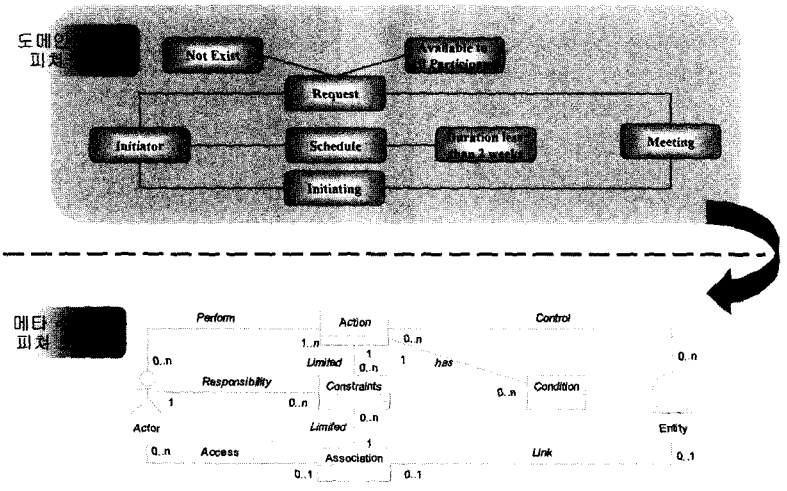

그림 3 UML로 표기한 메타 수준의 피처 모델

3.3 피쳐의 정형화 명세

지금까지 앞의 절들을 통하여 피쳐를 정형화 명세하기 위해 사용되어지는 술어를 어떻게 정의하고 표현이 되어지는가를 알아봤다. 본 절에서는 이러한 과정을 통하여 얻어진 술어들을 사용하여 피쳐들의 구조화와 명세화 하는 방안에 대해서 제안하고자 한다.

3.3.1 피쳐의 범주화와 구조

FODA에서는 피쳐들을 크게 비기능적 환경, 기능적 그리고 표현적 피쳐로 구분하여 나타내고 있다[3,4]. 이러한 구분은 피쳐가 소프트웨어 시스템의 전반적인 임무 또는 비 기능적인 면들을 포함하고 있으며, 사용자에게 어떠한 기능을 제공하는지 그리고 실제 사용자에게 보여주는 면들을 포함하고 있기 때문이다. 각각의 범주들은 다음과 같이 정의되어지고 있다.

- 비기능적 환경 피쳐 : 시스템의 전반적인 임무 또는 사용 패턴에 대한 특징들을 말하며, 성능, 정확도 등의 품질에 관련된 비기능적인 면들을 포함한다.
- 기능적 피쳐 : 소프트웨어 시스템이 수행해야 하는 실질적인 기능들을 나타내는 피쳐들이다.
- 표현적 피쳐 : 소프트웨어 시스템이 가지고 있는 정보들이 어떠한 방식으로 사용자들 또는 다른 시스템에 보여지거나 제공되어 지는가에 대한 특징들을 나타낸다.

이렇게 구분되어진 피쳐들은 나타내고자 하는 의미를 보다 정확하게 표현하고자 다음과 같은 속성들을 가질 것을 제안한다.

- 이름 : 피쳐의 고유한 이름
- 타입 : 어떤 종류의 피쳐인지를 의미하는 요소로써, 특성, 운영 환경, 도메인 기술 그리고 구현 기술로 구분되어진다.
- 공통성 : 피쳐의 공유성을 나타내는 요소로써, 그 성격에 따라서 반드시 필요한 피쳐인 경우 '필수적', 선택이 가능한 경우는 '선택적', 일련의 피쳐들 중 반드시 하나만 존재해야 하는 경우는 '양자택일적' 피쳐로 구분되어진다.
- 범주 : 해당 피쳐가 어느 범주 상에 포함되어 있는지를 의미한다. 비기능적 환경, 기능적 그리고 표현적 피쳐로 구분되어진다.
- 관련 요소 : 피쳐의 행위를 모델링 했을 경우, 관련되어지는 여러 요소들을 의미한다. 즉, 앞에서 정의한 메타 수준의 피쳐 모델에서의 여러 가지 요소들을 의미한다. 예를 들면, 액션을 의미하는 Initiate, 엔티티를 의미하는 Meeting 등이 포함될 수 있다.
- 바인딩 시간 : 해당 피쳐가 시스템에 바인딩 되는 시점을 의미하는 요소이다. 바인딩 시점에 따라서 사용, 재사용 그리고 로딩 시간으로 구분되어진다.
- 속성 : 피쳐의 속성을 나타내며, 피쳐의 상호작용 관리 시에 휴리스틱한 방법에서 사용되어진다. 그 속성에 따라서 피쳐가 기능이나 무언가를 제공하여 주는 속성을 가지고 있으면 '지원', 시스템의 성능이나 기능을 제한하는 속성이면 '제한', 제공하지 않아야 할 속성이라면 '회피', 성능의 최적화를 위한 속성이면 '최적화'로 구분되어 질 수 있다.
- 중요도 : 피쳐의 중요도를 나타내는 요소로써, 0에서 1의 수치를 가지며, 피쳐들 간의 상호작용 발생 시 우선 순위에 따라서 해결책을 모색할 수 있다.
- 비정형명세 : 피쳐의 비 정형적인 명세를 의미한다. 자연어로 피쳐의 의미하는 바를 기술한다.
- 정형명세 : 메타 수준에서 모델링 한 여러 요소들과 Z 그리고 KAOS에서 제공하는 Temporal Logic을 사용하여 X ⇒ Y의 implies 의 형태로 표기한다. 이러한 정형적인 명세는 피쳐의 의미를 정확하게 해주며, 기존에 상호작용 관리에 사용되었던 시스템적 기술들을 사용가능 하도록 지원해준다.

기본적인 피쳐의 속성들 이외에 각 범주의 피쳐들 만이 가지는 특징적인 속성들이 추가적으로 존재한다. 추가적인 속성들은 비기능적 환경, 기능적, 표현적 피쳐를 명세하기에 필요한 사항들이다. 각 범주의 피쳐들은 의미를 잘 전달하기 위하여 속성을 가지고 있다.

비기능적 환경 피쳐는 비 기능적인 요소가 영향을 미치는 다른 피쳐나 액터 또는 기타 메타 수준의 피쳐 모델에서 제시된 다른 요소들을 의미하는 영향요소와 어떠한 액터에 의해서 해당 피쳐가 제공되어지는지를 나타내는 수행객체를 속성으로 가진다. 기능적 피쳐는 기능적인 면을 보여주는 피쳐가 실행되기 전의 조건을 나타내는 사전 조건과 해당 피쳐가 수행된 뒤의 여러 조건들의 변화된 상태를 나타내는 사후조건을 속성으로 가진다. 표현적 피쳐는 해당 피쳐가 어떠한 액터에게 보여주는 지를 나타내는 대상객체와 시스템의 어떠한 액터가 피쳐에 의해 보여지게 되는 정보를 소유하고 있는 지를 의미하는 정보속성을 가지고 있다.

지금까지 살펴본 피쳐의 기본적인 구조와 그들을 나타내는 여러 속성들은 UML에 의해서 표기되어질 수 있다. 그림 4에서 보는 바와 같이 피쳐는 속성들을 가지고 있고, 이러한 속성들을 비기능적 환경, 기능적, 표현적 피쳐가 상속을 받는다. 각 피쳐는 상속받은 속성 이외에 해당 범주의 피쳐에 필요한 속성들을 추가로 가진다.

이렇게 표현된 피쳐의 구조 명세는 확장성과 이식성, 구조적 표현이 용이하고, 웹 기반의 지원이 충분히 이루어지고 있는 XML을 사용하여 표현할 수 있다. 표 5는 기본적인 피쳐의 구조를 XML을 이용하여 표현하고 있다.

3.3.2 정형화 방법을 사용한 피쳐 표기

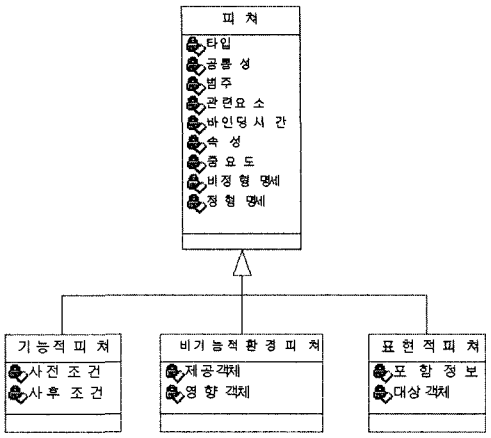

그림 4 UML로 표기한 피쳐의 구조

표 5 XML에 의한 기본 피쳐 구조의 명세

```
<?xml version="1.0" encoding="UTF-8" ?>
<!? Define Feature Structure -->
<!DOCTYPE FEATUREINFORMATIONS
[
<!ELEMENT FEATUREINFORMATIONS (Feature*)>
<!ELEMENT Feature (type, commonality, instanceOf,
concerns, bindingTime, attribute, informalDef,
formalDef, priority)>
<!ATTLIST Feature name CDATA #REQUIRED>
<!ELEMENT type (#PCDATA)>
<!ELEMENT commonality (#PCDATA)>
<!ELEMENT instanceOf (#PCDATA)>
<!ELEMENT concerns (#PCDATA)>
<!ELEMENT bindingTime (#PCDATA)>
<!ELEMENT attribute (#PCDATA)>
<!ELEMENT informalDef (#PCDATA)>
<!ELEMENT formalDef (#PCDATA)>
<!ELEMENT priority (#PCDATA)>
]
>
<FEATUREINFORMATIONS>
```

기존 피쳐 모델에서 발생하는 모호성, 이해의 오류 등의 문제들을 해결하기 위해서는 앞 절에서 제시한 피쳐의 구조화와 병행해서 피쳐에 대한 정형화 방법에 의한 표기가 이루어져야 한다. 따라서 본 절에서는 정형화 방법을 사용한 피쳐의 표기에 대하여 설명하고자 한다.

피쳐의 정형화 명세 언어로는 시스템의 기능적인 요소와 비기능적인 요소를 모두 표현할 수 있도록 Z와 실시간 Temporal Logic을 사용하였다. 이러한 Temporal Logic은 ERAE[18] 등에서 사용되어졌다. 본 논문에서 사용할 Temporal Logic은 기본적으로 사용되어져 왔던 연산자들을 사용할 것이다. 다음의 표 6은 본 논문에서

표 6 Temporal Logic 연산자

P : 현재의 상태
◆P : 일정 시간 전의 과거의 상태
◇P : 일정 시간 후의 미래의 상태
●P : 바로 전 단계의 상태
○P : 바로 다음 단계의 상태
■P : 언제나 과거인 상태
□P : 언제나 미래인 상태
WP : P를 제외한 언제나 미래의 상태
UP : P까지의 언제나 미래의 상태

사용되어질 연산자들을 보여주고 있다.

실제 시스템에서는 많은 제한 사항들이 존재하고 이러한 실시간의 제한을 나타내기 위해서는 추가적인 연산자들이 추가되어야 한다. 본 논문에서는 다음과 같은 [19]에서 소개되어진 연산자를 사용하도록 한다.

◇≤d : 제한 시간 d 이내의 미래의 어느 시간

□≤d : 제한 시간 d 까지의 모든 미래

피쳐의 정형화 명세의 기본적인 형식은 X ⇒ Y의 implies 형식을 취하도록 하였다.

표 7 피쳐를 정형화 명세 언어로 표기

Name : 미팅룸사용기간제한 InformalDef : 미팅 룸의 가용성을 보장하기 위해서 미팅 룸에 대한 사용 기간을 d로 제한한다. FormalDef : ∀p: Participant, r: Room Using(p, r) ⇒ ◇≤dUsing(p, r)
Name : 미팅초기화 InformalDef : 시스템은 Initiator가 미팅을 초기화 할 수 있는 기능을 제공해야 한다. FormalDef : (∀I : Initiator, p : Participant, m : Meeting) Request(I, m) ⇒ ◇(Schedule(m) ∨ Feasible(m)) ∧ Invited (p, m) ⇒ ◇Know(p, m)

표 7은 앞에서 살펴본 형식에 맞게 피쳐를 정형화 명세 언어로 명세한 예를 보여주고 있다. '미팅초기화' 피쳐에서 보는 바와 같이 피쳐는 기본적으로 X ⇒ Y의 implies의 형식을 취하고 있다. 비 정형적인 명세에서 간과되어질 수 있는 사항들 예를 들면, 요청이 이루어진 뒤에 스케줄에 미팅이 포함되어지거나, 확정된 미팅의 참가자들에 대한 초대 등의 명확한 명세가 정형화된 명세에서는 이루어지고 있음을 알 수 있다.

3.4 피쳐 정형화 명세 프로세스

피쳐 명세화 프로세스는 총 다섯 단계로 구분되어진다. 각 단계는 피쳐를 정형화 명세하기 위한 단계를 설

명하고 있다. 즉, 기존의 비 정형적인 피쳐에 대한 명세를 본 논문에서 제시하고 있는 정형화 명세 방법에 따라 명세하기 위한 방법을 피쳐 명세화 프로세스에서는 정의하고 있다. 피쳐 분석가에 의해 습득된 초기 피쳐들을 명세화 하기 위해서는 정형화에 사용되어질 술어들이 제공되어져야 하며, 술어들은 메타 수준의 추상화를 통하여 얻어질 수 있다. 술어들의 사전을 정의함으로써, 피쳐 분석가들은 일관성을 유지하며 명세화 작업을 수행할 수 있다. 또한, 술어를 정의하고, 피쳐를 명세화 하면서 발생되는 추가적인 사항들을 초기 피쳐 모델에 반영되어져야만 한다. 피쳐 명세화 프로세스에서는 이와 같은 사항들을 단계별로 정의하고 있다.

본 논문에서 제시하는 피쳐 정형화 명세는 그림 6과 같은 프로세스를 통해서 이루어진다.

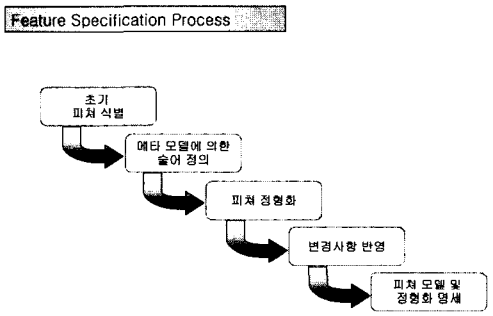

그림 5 피쳐 정형화 명세 프로세스

그림 5에서 보여주고 있는 피쳐 명세화 프로세스의 각 단계가 의미하는 바를 살펴보면 다음과 같다.

3.4.1 초기 피쳐 식별 단계

기존의 피쳐를 식별하고 비 정형적인 명세로 나타내는 단계이다. 피쳐 분석가는 사용자 설명서, 요구사항 문서, 소스코드 등으로부터 시스템의 피쳐들을 대략적으로 추출한 뒤, 이를 이용하여 피쳐 그래프를 작성하고, 자연어로 초기 피쳐에 대한 명세를 한다. 이 단계에서 추출되어진 피쳐는 초기 상태의 피쳐로써, 이후 명세화 과정을 통해서 추가적으로 새로운 피쳐가 추가되거나 세분화되어질 수 있다.

피쳐의 정형화 명세를 위하여 개발하고자 하는 시스템의 모든 기능 및 비 기능적인 요소들이 피쳐로써 습득되었는지를 확인하고, 정제화 과정을 거쳐 피쳐 그래프를 완성한다.

3.4.2 메타 모델에 의한 술어 정의

초기 피쳐 식별 단계를 통해 시스템의 피쳐들을 식별한 뒤, 식별된 피쳐들을 본 논문에서 제시하고 있는 메타 모델에 따라 추상화하여 모델링한다. 메타 모델에서 제시하고 있는 요소들, Actor, Action, Constraints, Condition, Association, Entity들을 정의하는 단계이다. 이 단계에서 정의되어진 각 요소들은 이후 피쳐의 정형화 명세에 사용되어질 술어로써 사용되어진다. 이렇듯 메타 모델에 따라서 명세화에 사용되는 술어를 정의하는 이유는 분석가에 따라서 서로 다른 술어를 사용하여 명세할 수 있기 때문이다. 동일한 피쳐를 서로 다른 술어를 사용하여 명세화 한다면, 정형화를 함으로써 얻을 수 있는 장점들을 얻을 수가 없다.

3.4.3 피쳐 정형화

피쳐의 정형화 명세에 사용되어질 술어들을 정의한 뒤, 제안하는 명세화 기법을 사용하여 정형화 명세를 한다. 명세에 사용되는 연산자는 Z와 Temporal Logic를 사용하며, 기본적인 명세의 형태는 X ⇒ Y의 imply 형태를 취한다. 이는 피쳐 상호 작용에서 사용될 기술들을 적용하기 위해서이다.

이전 단계에서 정의되어진 술어 사전을 통해서 명세에 사용될 술어를 선택하여 정형화 명세를 한다. 술어 사전에서 필요한 술어를 선택함으로써, 서로 다른 술어를 사용함으로써 발생하는 명세의 오류를 줄이는 효과를 기대할 수 있다.

3.4.4. 변경사항 반영

피쳐를 정형화 명세하는 과정을 통하여 기존 피쳐들의 변경이나, 술어들의 추가, 변경 또는 삭제가 발생한다. 즉, 초기 피쳐의 비 정형적인 명세에 정형적인 명세를 추가함으로써, 기존 피쳐에 대한 분석이 실시되고, 자연어로 명세 시에 간과되었던 부분이 이 단계에서 추가됨으로써, 기존 피쳐 모델의 변경이 발생한다.

발생된 피쳐와 술어의 변경사항은 기존 피쳐의 명세화와 술어 사전에 반영이 되어져야만 한다. 이 과정을 통하여 피쳐 모델은 보다 명확하게 의미하는 바를 정형화 명세를 통하여 나타낼 수 있다.

3.4.5 피쳐 모델 및 정형화 명세

습득된 피쳐와 정형화 기법에 의해 명세 되어진 피쳐들은 본 논문에서 제시하고 있는 피쳐의 구조에 맞추어 명세화 되어져야 한다. 피쳐의 구조를 이루는 각 속성들은 해당 피쳐가 어떠한 기능 및 비 기능적 요소를 가지고 있는지를 표현할 수 있는 요소들이다. 따라서, 프로세스의 앞 단계에서 얻어진 피쳐와 정형화 부분 그리고 추가된 사항들을 피쳐의 구조에 맞추어 명세함으로써, 피쳐의 정형화 명세가 완성되게 된다.

피쳐의 구조에 맞추어 명세 되어진 피쳐 모델은 XML을 사용하여 표기되어진다. XML을 사용하여 표기되어짐으로써, XML의 장점인 확장성과 이식성, 구조적 표현의 용이성 그리고 웹 기반의 지원을 얻을 수 있다.

3.5 피쳐 상호 작용 관리

지금까지 피쳐의 구조와 명세 방안에 대해서 알아보

았다. 앞에서 제안한 이러한 명세 방안은 피쳐에 대한 상호 작용 관리를 가능하게 하며, 이는 피쳐가 정형화 언어를 통하여 명세 되었기 때문이다. 따라서 본 논문에서는 피쳐의 상호 작용 관리를 통해서 제안하는 명세 방안에 대한 타당성을 제시하고자 한다.

3.5.1 피쳐 상호 작용 관리의 정의

소프트웨어 시스템을 개발하는 데 있어서 요구 사항 분석 또는 목표 지향 분석 단계에서는 요구사항 또는 목표들 간의 상호 작용에 대한 연구가 많이 진행되어 왔다. 이러한 분야에서의 상호 작용의 관리라 함은 발생하는 문제들을 발견해 내고 이에 대한 해결 방안을 제시하는데 초점을 두고있다[13]. 따라서, 본 논문에서는 피쳐들 간의 상호 작용 관리를 다음과 같이 정의한다.

피쳐 상호 작용 관리는 소프트웨어 시스템을 구성하는 피쳐들 사이에서 발생할 수 있는 여러 문제점들을 식별하고, 관리하는 일련의 활동들을 의미한다.

3.5.2 피쳐에서의 상호 작용 문제 분류

피쳐에서 다루는 상호 작용 문제들은 다음의 표 8과 같이 분류되어진다.

표 8 피쳐 상호작용 문제의 분류

분 류	내 용
술어	시스템에서 발생할 수 있는 하나의 상황을 여러 개의 비슷한 의미의 술어로 표현되어 질 경우
해석	하나의 술어가 다른 여러 의미로 해석되어 지는 경우
구조	피쳐 모델에서 사용되어지는 여러 객체 및 관련 요소들이 다른 구조로 표현되어 지는 경우
충돌	하나 이상의 피쳐들 간에 발생하는 충돌로 피쳐들의 집합은 영역내에서 불일치 하거나, 피쳐들 중 어느 하나라도 제거되면 불일치 관계는 깨어지는 조건을 만족한다.
차이	충돌관계의 약한 형태로써 경계조건 = 참인 경우가 이에 해당한다. (경계조건 : 피쳐들이 불일치하게 만드는 특별한 조건)
경쟁	동일한 Universal quantifier Feature의 여러 개체들 간에 차이가 발생했을 경우
방해	하나의 피쳐를 만족하는데 방해가 되는 경우가 경계조건에서 발생했을 경우

3.5.3 피쳐에서의 상호 작용 문제 식별

분류되어진 피쳐의 상호 작용 문제에서 발생하는 문제점들은 목표 회귀, 패턴 그리고 휴리스틱한 기술들에 의해 식별되어질 수 있다. 이러한 기술들은 피쳐가 정형화되어졌다는 조건에서 사용되어질 수 있는 기술들이다.

목표 회귀 기술을 이용한 방법은 피쳐의 Assertion Ai에 대한 부정을 적용한 뒤 이에 대한 역추적을 수행하여 상호 작용 문제를 일으키는 조건을 찾아내는 방법이다. 다음 수식은 이러한 조건을 표현하고 있다.

$$\{Dom, B, \wedge_{j \neq i} Aj\} \vdash \neg Ai \quad (1)$$

주어진 어떤 피쳐의 Assertion Ai가 있고, Ai의 부정과 이로부터 영역 이론으로부터 회귀를 통해 역추적을 실시하여 상호 작용 문제들이 일어나는 경계조건을 나타내는 Aj를 얻어내는 것을 식 1에서 보여주고 있다.

다음으로는 패턴에 의한 피쳐의 상호 작용 문제를 식별해내는 방법이다. 앞에서 살펴본 피쳐의 Assertion과 영역 정형 명세의 반복적인 회귀을 통해서 여러 피쳐의 상호 작용 문제를 식별해 낼 때, 분석가는 반복적으로 발생되어지는 피쳐의 상호 작용 문제들을 더불어 식별해 낼 수 있다. 이러한 패턴을 발견해 냄으로써, 분석가는 매번 회귀를 통하지 않고, 패턴을 사용하여 상호 작용을 식별해 낼 수 있다.

본 논문에서는 목표 지향 분석에서 사용되었던 몇 가지 패턴을 피쳐에 적용하여 상호 작용 문제를 식별해 낼 수 있었다. 그림 6은 이러한 패턴들을 보여주고 있다.

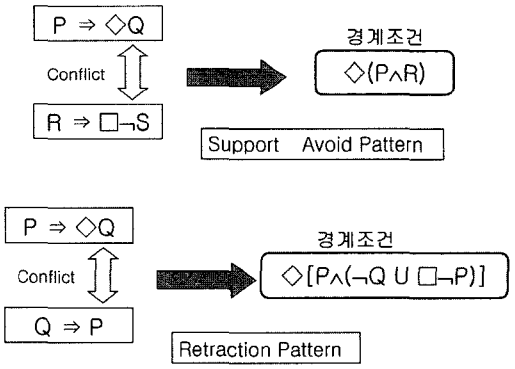

그림 6 피쳐 상호 작용 문제 식별 패턴

그림 6에서 첫 번 째 패턴은 피쳐의 속성으로써 '지원'과 '회피'를 가지는 피쳐들 사이에 발생하는 상호 작용 문제를 도출해낸다. 예를 들면 다음과 같다. 미팅의 사용자가 미팅에 필요한 자원들을 계속적으로 사용할 수 있도록 사용자의 사용성를 보장해주는 피쳐와 자원을 사용하되 믿을 수 없는 자원들에 대해서는 사용을 못하게 하는 피쳐가 존재한다고 하자. 이 두 가지 피쳐에 대한 정형화 명세는 다음과 같다.

Name : KeepUsing

...

FormalDef : ∀p : Participant, r : Resource

Requesting(p, r) ⇒ ◊ Using(p, r)

Name : AvoidUnreliableResource

...

FormalDef : ∀p : Participant, r : Resource

¬Reliable(r) ⇒ □¬Using(p, r)

위의 피쳐들의 정형화 명세에서 P: Requesting(p, r), Q: Using(p, r), R: ØReliable(r), S: Using(u, r) 이라고 하면, 패턴에 의해서 자동으로 경계 조건인

◊∃p : Participant, r : Resource

Requesting(u, r) ∧ ¬Reliable(r)

을 도출해낼 수 있다.

두 번째 패턴은 레스트릭션 패턴으로 하나의 피쳐가 다른 피쳐에 의해서 그 행동이 취소가 되어버리는 상호작용을 추출하는데 사용되어지는 패턴이다.

마지막으로, 피쳐의 상호 작용들은 휴리스틱한 방법을 통하여 식별되어질 수 있다. 이러한 휴리스틱한 방법들은 패턴 또는 과거의 경험에 의해서 얻어지는 것들이며 다음과 같은 예들이 있다. 첫째, SatisfactionFeature와 SaftyFeature가 존재 시 두 피쳐가 동일한 개체에 연관되어 있다면 상호 작용이 일어난다. 둘째, ConfidentialityFeature와 InformationFeature가 동일한 개체 정보에 관련되어 있다면 상호 작용이 발생한다.

3.5.4 피쳐 상호 작용 문제 해결

앞 절에서 식별된 피쳐의 상호 작용 문제들은 몇 가지 방법에 의하여 해결되어질 수 있다.

첫째, 경계 조건 회피이다. 충돌을 발생시키는 경계 조건이 일어나지 않도록 경계 조건에 대해서 부정을 하여 피하도록 하는 것이다. 다음과 같이 식 (2)로 표현되어질 수 있다.

$$P \Rightarrow \Box\neg Q \qquad (2)$$

둘째, 피쳐 복구이다. 경계조건은 언제나 피할 수 있는 것이 아니다. 즉, 외부의 환경이나 소프트웨어의 엑터의 관리 밖의 조건인 경우 경계 조건이 발생한 뒤, 일정 시간이 경과한 뒤에는 다시 그 경계 상황이 해소되어 복구가 이루어 질 때까지 대기하는 방법이다.

셋째, 피쳐의 약화이다. 상호작용이 일어나는 피쳐에 대해서 약화 작업을 함으로써, 상호작용이 발생하는 피쳐 중의 하나가 경계 조건이 발생하지 않도록 하는 조건을 포함하게 만든다.

넷째, 해결 패턴이 있다. 앞에서 언급되었던 여러 해결 방법들을 적용하면서 반복적으로 나타나는 해결책에 대해서 패턴화하여 적용하는 방법을 의미한다.

다섯째, 선택적 피쳐 세분화가 존재한다. 상호 작용이 발생 시, 상호 작용이 발생하는 피쳐에 대해서 상위 수준의 피쳐로 거슬러 올라가 상호 작용이 발생하지 않도록 다시 피쳐를 세분화하는 방법을 말한다. 즉, 선택적인 피쳐의 세분화를 고려하는 것이다.

4. 분산 미팅 스케줄러 시스템 예제의 적용

본 장에서는 앞서 제시한 피쳐의 정형화 명세 기법을 분산 미팅 스케줄러 시스템[20,21]의 예에 적용시켜보도록 한다. 영역에 대한 설명을 한 뒤, 도입 부분에서 언급했던 기존 피쳐 모델에서 발생했던 여러 가지 문제점들이 제안하는 피쳐 정형화 명세 기법을 통하여 어떻게 해결되어 졌는지를 보이도록 하겠다.

4.1 분산 미팅 스케줄러 시스템

분산 미팅 스케줄러 시스템[20][21]은 분산된 환경에서 미팅의 초기화 및 스케줄 관리 그리고 필요한 여러 가지 자원들을 효율적으로 관리하도록 하기 위한 시스템이다. 시스템은 사용하는 규모에 따라서 자원에 대한 관리, 회계 관리 등의 기능에 대한 포함 여부를 결정한다.

다음의 그림 7은 분산 미팅 스케줄러 영역에서의 전반적인 기능들을 보여주고 있다.

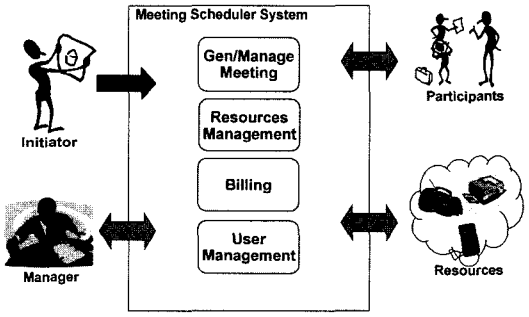

그림 7 분산 미팅 스케줄러 시스템

분산 미팅 스케줄러 시스템은 그림 7에서 보여주는 바와 같이 분산된 환경에서 미팅에 대한 관리를 지원하며, 관련 자원 및 사용자들에 대한 관리를 지원한다. 분산 미팅 스케줄러 시스템은 이미 요구 사항 분석 또는 목표 지향 분석 그리고 정형화 방법 분야에서 예제로 많이 다루어져서 영역에 대한 분석이 충분히 이루어져 있고 다른 방법들에 의해 분석이 되어있어 상대 비교가 가능하다. 또한, 이미 영역에 대한 조사가 충분히 이루어져 있기 때문에 발생 가능한 여러 상황들에 대한 테스트가 가능하다는 이점이 있다.

4.2 피쳐 정형화 명세를 통한 문제 해결

2장에서 기존 피쳐 모델의 비 정형성으로부터 발생하는 문제점을 분석해 보았다. 문제점들은 크게 이해의 오류, 의미의 모호성, 불 완전성 그리고 상호 작용 문제로 분류되어졌다. 다음은 문제점들이 피쳐의 정형화 명세를 통해 어떻게 해결되어지는지를 보여준다.

4.2.1 이해의 오류

이해의 오류는 서로 다른 의미를 가지는 두 피쳐가 비 정형적인 명세로 인하여 이해의 오류를 발생시켜 동일한 피쳐로써 이해가 되어지는 문제이다. 이러한 문제는 본 예제에서 'ChangeParticipant'와 'ModifyParticipant' 피쳐로부터 발생한다. 즉, 'Change'와 'Modify'가 가지는 단어적 비슷함으로 인하여 분석가의 의도와는 다르게 두 피쳐가 동일하게 인식이 되어 피쳐 모델에 따로 존재하여 서로 다른 기능을 보여줘야 하지만 그렇지 못한 경우이다. 표 9는 두 피쳐의 명세를 정형화를 통해서 나타냄으로써, 이해의 오류를 해결하였다.

표 9 ChangeParticipant와 MocifyParticipant 피쳐의 정형화 명세

Feature[ChangeParticipant] FormalDef (∀nP:newParticipant, dP:deleteParticipant, m:Meeting) (curDate < m.StartDate) ∧ (nP.availableDate I m.Duration) ∧ OnMeetingList(m.dP) ⇒(AddParticipant(m, nP)∧ (DelParticipant(m.dP))
Feature[ModifyParticipant] FormalDef (∀i : initiator, p:participant, m:meeting, nP:newParticipant) (m.initiator = I) ∧ onParticipantList(m, p) ⇒ ChangeParticipant(p)

두 피쳐는 표 9에서와 같은 명세를 통하여, 'ChangeParticipant'는 미팅에 참가하는 참가자들을 변경하는 피쳐이고 'ModifyParticipant'는 참가자 개개인의 신상명세를 수정하는 피쳐 임이 구분되어 서로 다른 피쳐 임을 표현할 수 있다

피쳐 정형화 명세 공정을 거치며 'ModifyParticipant'가 애플리케이션마다 필요한 공통된 피쳐와 선택 가능한 피쳐로 보다 세분화 되어야할 필요성이 도출되었다. 따라서, 최초 '필수적'이던 피쳐는 정형화 공정을 통하여 'ModifyParticipantGenInfo'와 'ModifyParticipantSpecialInfo'로 세분화되어, '필수적'과 '선택적' 피쳐로 나뉘어졌다. 즉, 정형화 명세과정을 거쳐 가장 세분화된 피쳐들을 여러 애플리케이션에 사용될 수 있는 재사용 가능한 공통점과 해당 애플리케이션에만 사용될 수 있는 차이점으로 분류할 수 있었다.

그림 8은 'ChangeParticipant' 피쳐와 'ModifyParticipant' 피쳐 간의 이해의 오류 문제를 해결한 뒤의 피쳐 모델을 AND/OR 그래프의 형태로 나타낸 것이다. 그림 8에서 보는 바와 같이 두 피쳐는 서로 다른 피쳐로 인식되어 피쳐 모델에 동시에 존재하고 있다.

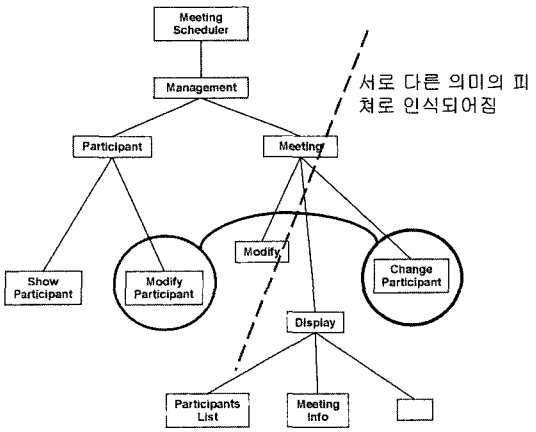

그림 8 이해의 오류 해결 뒤의 피쳐 모델

4.2.2 모호성

피쳐 분석가에 따라서 동일한 의미를 가지는 피쳐가 다른 표기법을 가지고 명세 되어질 수 있다. 이때 의미가 모호한 표기법을 사용한다면 동일한 의미의 피쳐 임에도 불구하고, 서로 다른 피쳐로 인식되어 피쳐 모델에 동시에 존재하는 중복성 문제를 야기시킨다. 본 예제에서 이러한 문제는 'GenerateMeeting'과 'InitiateMeeting'의 두 피쳐 사이에 발생한다. 미팅을 초기화시키는 것과 관련된 피쳐를 분석해 내었다고 했을 때, 분석가들은 자신의 취향에 따라서 다른 술어를 사용하여 표현할 수 있을 것이다. 이 경우 두 피쳐는 서로 다른 술어를 사용하여 다른 피쳐로 인식이 되어질 수 있다. 사용되어지는 술어가 가지는 모호성으로 인하여 발생하는 문제인 것이다. 하지만, 제안하는 명세화 방법을 통하여 정형화 명세를 할 경우 두 피쳐는 동일한 피쳐를 나타내므로 다음과 같은 명세를 통하여 동일한 피쳐 임을 표현하게 되고, 이를 통하여 피쳐의 모호성을 해결한다.

표 10 InitiateMeeting과 GenerateMeeting의 동일한 명세

Feature[InitiateMeeting] FormalDef (∀i initiator, p:Participant, m:Meeting) Request(I, m) ⇒ ◇(Schedule(m) ∧ ¬ Feasible(m)) ∧ Invited(p, m) ⇒ ◇Know(p, m)
Feature[GenerateMeeting] FormalDef (∀i initiator, p:Participant, m:Meeting) Request(I, m) ⇒ ◇(Schedule(m) ∧ ¬ Feasible(m)) ∧ Invited(p, m) ⇒ ◇Know(p, m)

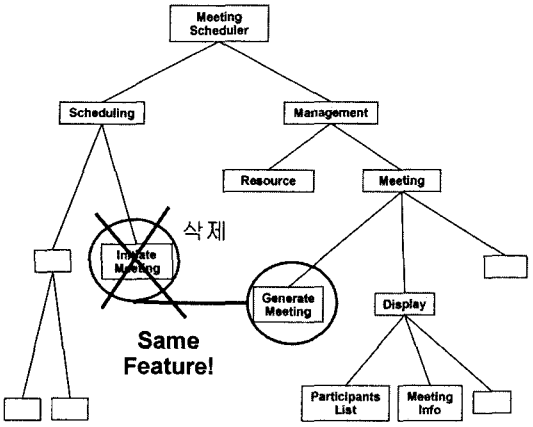

그림 9 모호성이 해결된 후의 피쳐 모델

표 10은 두 피쳐의 명세를 정형화를 통해서 나타냄으로써, 모호성을 해결한 것을 보여주고 있다. 즉, 다른 술어를 사용했지만 정형화 명세를 거침으로 인하여 두 피쳐가 동일한 피쳐라는 것을 나타낼 수 있다.

그림 9는 명세화 과정을 통해서 기존의 피쳐 모델로부터 중복되는 피쳐가 제거되는 모습을 보여주고 있다. 그림 9에서와 같이 모호성이 해결됨으로 인하여 기존 피쳐 모델에 존재하는 'InitiateMeeting' 피쳐가 삭제되었다.

4.2.3 불완전성(Incompleteness)

비 정형적인 피쳐의 명세로 인하여 충분한 기능이 명세에 포함되어야 함에도 불구하고 명세의 부재 혹은 모순되는 명세로 원하지 않는 기능의 포함 또는 부재가 발생할 수 있다. 본 예제에서는 'ExtendMeetingDate' 피쳐로부터 그 예를 찾아볼 수 있다. 이 피쳐는 미팅이 진행되는 중 필요에 의해 그 기간을 연장할 수 있는 피쳐이다. 하지만, 이 피쳐는 분석가의 판단에 따라서 'ModifyMeeting'이라는 미팅의 속성을 변경할 수 있는 피쳐에 포함되어질 수도 있다. 이 경우 잘못된 해석의 오류로 인하여, 구현되는 시스템에 필요한 기능이던 'ExtendMeetingDate' 기능이 빠지는 불완전한 시스템을 만들게 된다.

표 11은 기존의 FODA에서 'ExtendMeetingDate' 피쳐의 비 정형적인 명세법과 제안하는 방법에서의 정형화 명세를 비교를 통하여 보여주고 있다. 정형화 명세를 통하여 'ExtendMeetingDate'이 특성에 관련된 피쳐이며, 필수적이고, 우선순위가 1로 분석되어지므로 상호작용 발생 시에 먼저 고려대상이 되어지며, 관련요소들을 통하여 추후 변경 발생 시 영향을 받거나 고려해야할 객체를 분석할 수 있다. 제안하는 프로세스에 따라서 'ExtendMeetingDate' 피쳐를 분석하고 정형화 함으로써, 'ModifyMeeting' 피쳐에 종속적인 피쳐가 아닌 독립적인 피쳐로써 애플리케이션에 포함되어져야 한다는 것을 분석해 낼 수 있었다.

표 11 기존의 명세 방법과 제안하는 방법의 비교

기존 FODA에서의 표기법
Name : ExtendMeetingDate Description : Extend the duration of the Meeting Date
제안하는 방법에서의 명세
Name : ExtendMeetingDate Type : Capability Commonality : Mandatory InstanceOf :Operational Concerns : ExtendMeeting, Initiator, Meeting, Duration BindingTime : Usetime Attribute : Support Priority : 1 .InformalDef : Extend the duration of the Meeting Date FormalDef : (∀i initiator, m: Meeting, nD: newDuration) IsRunning(m) ∧ Initiator(I, m) ⇒ ○Extend(m, nD)

4.2.4 피쳐 상호 작용 문제

소프트웨어 시스템이 개발된 이후에 런타임 시에 피쳐들 간의 예상치 못한 충돌이 발생할 수 있는데 이러한 문제는 피쳐 상호 작용 관리를 통하여 해결되어질 수 있다. 앞 절에서 살펴보았듯이 피쳐 상호 작용 관리는 상호 작용 식별과 해결 방안으로 구분되어지며, 이 과정에서 기존에 사용되어지던 목표 회귀, 패턴, 그리고 휴리스틱한 기술들을 사용하였으며, 이는 피쳐의 정형화 명세로써 가능하였다.

사용자들이 미팅 룸을 계속적으로 사용하기를 원할 경우 사용하도록 하는 'UseRoomAsUserNeed' 피쳐와 미팅 룸의 가용성을 위해서 미팅 룸을 제공 시 일정 기간의 제한을 두는 'KeepRoomAvailable' 피쳐는 '사용자가 제한된 기간 이상으로 미팅 룸이 필요할 경우'의 경계 조건(boundary condition)에서 충돌이 일어난다. 이러한 상호 작용에 의한 충돌은 회귀를 통한 역 추적을 통하여 식별이 가능하였다.

'UseRoomAsUserNeed'와 'KeepRoomAvailable' 피쳐의 정형화 명세는 다음과 같다.

∀u: User, r: Room

Using(u, r) ⇒ ○[Needs(u, r) → Using(u, r)]

· · · · · · UseRoomAsUserNeed 명세

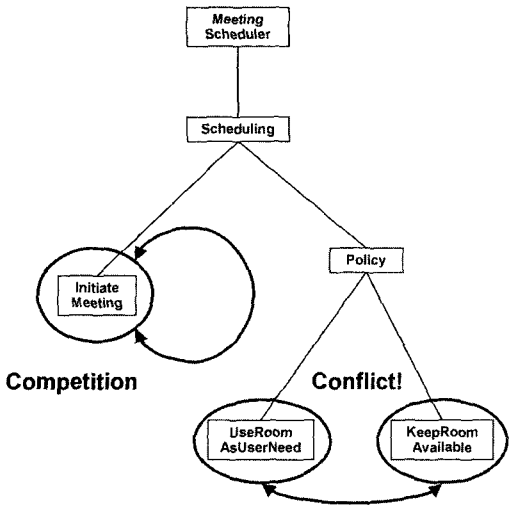

그림 10 상호 작용이 식별되어진 후의 피쳐 모델

∀u: User, r: Room

Using(u, r) ⇒ ◇≤dUsing(u, r)

· · · · · · KeepRoomAvailable 명세

다음은 회귀를 통해서 얻어낸 경계 조건이다.

◇(∃u: User, r:Resource)[Using(u, r)∧□≤ dNeeds(u, r)]

· · · · · · 두 피쳐의 경계 조건

또한, 휴리스틱한 방법을 통하여 'InitiateMeeting' 피쳐가 동일한 날짜에 하나 이상의 미팅을 초기화하려고 하는 경우에 경쟁 상호 작용이 발생하는 것을 식별해 낼 수 있다. 그림 10은 앞에서 살펴본 예를 피쳐 모델에서 보여주고 있다.

기존 피쳐 모델에서는 발견할 수 없었던 충돌 또는 경쟁의 상호 작용 문제가 목표 회귀, 휴리스틱한 방법들로 식별되어 피쳐 모델에 반영이 된 것을 그림 10은 보여주고 있다.

5. 결론 및 향후 연구 과제

점점 더 크고, 복잡해지는 소프트웨어와 사용자의 요구가 급변하는 시장에 대응하고자 소프트웨어 재사용에 대한 관심이 더욱 높아지고 있다. 특히, 프로덕트 라인 개발은 특정 도메인 내에서의 재사용을 극대화하려는 개발 방법으로서 최근 SEI를 중심으로 그 연구가 활발히 진행되어지고 있다.

따라서 본 논문에서는 프로덕트 라인 개발 방법에서 특정 도메인 내의 여러 애플리케이션간의 공통점과 차이점을 도출해내는 방법으로 사용되어지는 피쳐에 대한 정형화된 명세 방법을 제시함으로써 피쳐에 대한 비 정형적 명세로부터 발생하는 모호성, 이해의 오류, 불 완전성, 그리고 상호 작용 문제 등에 대한 해결을 가능하게 하였다. 이 과정에서 피쳐의 행동을 메타 수준에서 모델링하여 영역에 독립적인 피쳐의 구조와 언어를 정의하였으며, KAOS에서 제공되어지는 다중 패러다임 언어를 사용하여, 피쳐와 여러 메타 모델을 정형화하여 명세화 하는 방법과 이러한 요소들을 습득하는 프로세스를 제시하였다. 그리고 앞에서 제시된 정형화된 피쳐 명세를 피쳐의 상호 작용 관리와 분산 미팅 스케줄러 시스템에 적용하여 여러 문제들을 해결해 봄으로써 본 논문에서 제안하는 방법에 대한 타당성을 제시하였다.

아울러 향후에는 이러한 연구를 좀 더 보안하여, 프로덕트 라인 개발 방법에서 사용되어지는 비즈니스 전략적인 측면이 추가되어진 확장된 명세화 기법에 대한 연구가 진행되어야 하겠다. 피쳐의 구조와 명세기법이 정형화 되어진다면, 피쳐 상호작용 관리의 자동화된 분석이 가능하므로 피쳐들 간에 발생되는 여러 상호작용을 자동적으로 검출해내는 연구가 가능할 것이다. 또한, 미팅 스케줄러 시스템에 적용되었던 적용 범위를 보다 다양하고 커다란 영역에 적용함으로써, 제안하는 명세화 방법을 보완하는 연구도 진행되어야 할 것이다.

참 고 문 헌

[1] P.Clements and L.Northrop, Software Product Lines: Practices and Patterns, Boston, MA: Addision Wesley Longman, Inc., 2001.

[2] P.Donohoe, Software Product Lines: Experience and Research Directions, Denver, Colorado, Boston, MA: Kluwer Academic Publishers, pp. 28-31, August 2000.

[3] C.Kang, S.G.Cohen, J.A Hess, W.E.Novak, and A.S.Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, Technical Report CMU/SEI-90-TR-21, Pittsburgh, PA, Software Engineering Institute, Carnegie Mellon University, 1990.

[4] Cohen, G.Sholom, Jr.Stanley, L.Jay, Peterson, S. Spencer & Jr. Krut, W.Robert, Application of Feature-Oriented Domain Analysis to the Army Movement Control Domain (CMU/SEI-91-TR-28). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1992.

[5] A.Dardenne, A.van Lamsweerde and S.Fickas, Goal-Directed Requirements Acquisition, Science of Computer Programming, Vol.20, pp. 3-50. 1993.

[6] A.van Lamsweerde, R.Darimont and E.Letier, Managing Conflicts in Goal-Driven Requirements Engineering, IEEE Trans. On Software. Engineering, Special Issue on Inconsistency Management in Software Development, November 1998.

[7] C.Kang, S.Kim, J.J.Lee, K.Kim, E.Shin, M.Huh, FORM: A Feature-Oriented Reuse Method with

Domain-Specific Reference Architectures, Annals of Software Engineering, 5: pp.143-168, 1998.
[8] A.D.Vici and N.Argentieri, FODAcom: An Experience with Domain Analysis in the Italian Telecom Industry, Proceedings of the 5th International Conference on Software Reuse (ICSR), Victoria, BC, Canada, pp.166-175, June 1998.
[9] M.L.Griss, J.Favaro and M.d' Alessandro, Integrating Feature Modeling with the RSEB, Proceedings of the 5th International Conference on Software Reuse (I, Victoria, BC, Canada, pp.76-85, June 1998.
[10] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kwanwoo Lee, Feature-Oriented Engineering of PBX Software for Adaptability and Reusability, Software Practice & Experience, Vol. 29 No.10, pp. 875-896, 1999.
[11] A. van Lamsweerde, Formal Specification: a Roadmap. In The Future of Software Engineering, A. Finkelstein(ed.), ACM Press, 2000.
[12] Martin Fowler and Kendall Scott, UML Distilled 2nd Edition, Addison Wesley, 2000.
[13] W.N.Robinson, S.Pawlowski, S.Volkov, Requirements Interaction Management, GSU CIS Working Paper 99-7, Georgia State University, Atlanta, GA, August 1999.
[14] B.Potter, J.Sinclair and D.Till, An Introduction to Formal Specification and Z. Second edition, Prentice Hall, 1996.
[15] J.M. spivey, The Z Notation, Prentice Hall, 1989.
[16] C.B. Jones, Systematic Software Development Using VDM, 2nd Edition, Prentice Hall, 1990.
[17] XML Specification, http://www.w3c.org/TR/WD-xml.html
[18] E.Dubois, J.Hagelstein and A.Rifaut, A Formal Language for the Requirements Engineering of Computer Systems, Introducing a Logic Based Approach to Artificial Intelligence, A. Thayse(Ed.), Vol. 3, Wiley, pp.357-433, 1991.
[19] R.Koyamans, Specifying Message Passing and Time-Critical Systems with Temporal Logic. Springer-Verlag, 1992.
[20] M.S. Feather, S. Fickas, A. Finkelstein, A. van Lamsweerde, Requirements and Specification Exemplars Automated Software Engineering, Kluwer Pubs, Vol. 4, No. 4 1997.
[21] A. van Lamsweerde, R. Darimont and Ph. Massonet, Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt, Proceedings RE'95-Second International Conference on Requirements Engineering (York, UK), IEEE Computer Society Press, March. pp. 194-203. 1995,

송 재 승
2000년 서강대학교 컴퓨터학과 공학학사
2002년 서강대학교 컴퓨터학과 공학석사
2002년~현재 LG전자 CDMA단말연구소 연구원. 관심분야는 요구공학, 소프트웨어 프로덕트 라인, 임베디드 소프트웨어

김 민 성
2001년 서강대학교 컴퓨터학과 공학학사
2003년 서강대학교 컴퓨터학과 공학석사
2003년 현재 서강대학교 컴퓨터학과 박사과정. 관심분야는 요구공학, 소프트웨어 프로덕트 라인, 소프트웨어 아키텍처

박 수 용
정보과학회논문지 : 소프트웨어 및 응용 제 30 권 제 10 호 참조