컴포넌트 기반 소프트웨어 개발의 효율적인 재사용성을 지원하기 위한 도메인 설계 방법

(Domain Design Method to Support Effective Reuse in Component-Based Software Development)

문 미 경 † 박 준 석 † 염 근 혁 ††
(Mikyeong Moon) (Joonseok Park) (Keunhyuk Yeom)

요 약 컴포넌트와 재사용의 개념을 함께 가지는 컴포넌트 기반 소프트웨어 개발 방법은 소프트웨어 개발시간과 비용을 줄이고, 생산성을 향상시키는 등의 장점을 가진다. 이때 컴포넌트들의 재사용을 체계적으로 지원하기 위해 컴포넌트 기반 소프트웨어 개발 프로세스와 병행한 도메인 분석과 설계 방법이 필요하다. 또한 현재의 도메인 분석, 설계 방법에서 부족한 도메인 내의 공통성과 다양성을 인식하는 과정에 대해 객관적인 분석 제시가 필요하다. 그리고 그 정보를 도메인 모델에 잘 반영시켜 그로부터 도메인 컴포넌트를 추출하고 이를 도메인 아키텍처에 명시적으로 나타내는 방법이 필요하다. 본 논문에서는 컴포넌트 기반 소프트웨어 개발 시 효율적인 재사용 방안으로 도메인을 체계적으로 정의하고 분석, 설계하는 방법을 제안한다. 도메인 내에서 다시 사용될 수 있는 부분, 즉 공통성을 가진 요소들을 요구사항 분석 단계에서부터 추출하여 이를 계속 유지, 정제시켜가며 각 단계의 산출물에 반영시킨다. 이 과정을 통해 공통성을 가진 형태의 도메인 컴포넌트를 생성해 낼 수 있으며 이를 기반으로 도메인 아키텍처를 설계할 수 있다. 본 논문에서 제시하는 도메인 분석, 설계를 통해 소프트웨어를 재사용(software reuse)할 수 있도록 해 주는 동시에, 재사용 가능한 소프트웨어(reusable software)를 생산할 수 있도록 함으로써 체계적인 재사용의 선순환 관계를 지원하게 된다.

키워드 : 도메인 분석, 도메인 설계, 도메인 공학, 컴포넌트 기반 소프트웨어 개발

Abstract Component-based Software Development(CBSD) supported by both component and reusability can reduce development time and cost, and also can achieve high productivity. To support component reusability systematically domain analysis and design in parallel with CBSD-process is needed. And also it is needed to suggest objective analysis process to fine out commonality and variability in domain, which is lacked in current domain analysis and design method. And to abstract domain component from the information which is well reflected in domain model, and to express it in domain architecture is needed. In this paper, we suggest the method to define, analyze and design domain systematically for enhancing reusability effectively in Component-base Software Development. We abstract components which can be reusable in domain, in other word, which have commonality from requirement analysis level. We sustain and refine them. And we reflect them to the products of each level. From these process, we can produce the domain component which have commonality. On this basis, we can design domain architecture. In this paper, to produce reusable software we investigate new systematic approach to domain analysis and design from the view point of software reusability.

Key words : Domain analysis, Domain Design, Domain Engineering, Component-based Software Development

· 본 연구는 한국과학재단 목적기초연구(R05-2000-000-00276-0) 지원으로 수행되었음.

† 비 회 원 : 부산대학교 컴퓨터공학과
mkmoon@pusan.ac.kr
pjs50@pusan.ac.kr
†† 종신회원 : 부산대학교 컴퓨터공학과 교수
yeom@pusan.ac.kr

논문접수 : 2002년 6월 24일
심사완료 : 2003년 2월 12일

1. 서 론

소프트웨어 시스템 세계에서 '도메인'이란 용어는 특정 분야에서 공통의 기능을 가지는 일련의 시스템의 집합을 나타내는 것이다. 예를 들어, 의료 관련 소프트웨어 시스템의 도메인에서는 "의사가 진료하는 것", "환자가 수속을 밟는 것"과 같이 객체와 행위들로 이루어져서 도메인 요소가 된다[1]. 이와 같이 특정 도메인 내에서 공통성과 다양성을 찾아내고 이들을 다시 재사용 될 수 있는 형태로 만들 수 있다면 그 도메인 내에 속하는 또 다른 새로운 시스템을 만들 때 이를 유용한 정보로 이용할 수 있게 된다. 이는 시스템 개발을 용이하게 하고 결과적으로 생산성을 높이고 개발 시간과 비용을 줄일 수 있게 해준다. 또한 이는 오늘날과 같은 복잡하고 대형의 컴퓨터 시스템들을 매우 짧은 시간 내에 개발하고자 할 때 더욱 요구되는 사항이다. 하지만 하드웨어의 재사용성과는 달리 소프트웨어는 많은 다양성을 가지고 있어 이를 식별하고 표준화시키는 것은 어려운 작업이지만, 컴포넌트 기반 개발이라는 새로운 패러다임을 만남으로써 그 발전의 가능성을 보여주고 있다.

지금까지 연구되어진 도메인 분석, 설계 방법들은 기존의 전통적인 개발 프로세스 즉, 분석, 설계, 구현의 과정에 적합한 것이다. 따라서 컴포넌트 기반 소프트웨어 개발 프로세스에 병행하여 컴포넌트들의 재사용을 체계적으로 지원하기 위한 진보된 도메인 분석과 설계 방법의 필요성이 요구된다. 또한 현재의 도메인 분석, 설계 방법은 도메인 내의 공통성과 다양성을 결정하는 과정에 대해 구체적인 분석 틀을 제시하지 못하고 있으며, 그 정보를 도메인 모델에 반영시켜 그로부터 도메인 컴포넌트를 추출하고 이를 도메인 아키텍처에 명시적으로 나타내는 방법이 부족하다.

본 논문에서는 컴포넌트 기반 소프트웨어 개발 시 효율적인 재사용 방안으로 도메인을 체계적으로 정의하고 분석, 설계하는 방법을 제안한다. 도메인 내에서 다시 사용될 수 있는 부분, 즉 공통성을 가진 요소들을 요구사항 분석 단계에서부터 추출하여 이를 계속 유지, 정제시켜가며 각 단계의 산출물에 반영시킨다. 이 과정을 통해 공통성을 가진 형태의 도메인 컴포넌트를 생성해 낼 수 있으며 이를 기반으로 도메인 아키텍처를 설계할 수 있다. 공통성과 다양성의 속성을 가진 도메인 아키텍처는 도메인에 속한 여러 시스템에서 사용 될 수 있고, 각 시스템의 다양한 특성을 반영할 수 있는 유연성을 가지게 된다. 본 논문에서 제시하는 도메인 분석, 설계를 통해 소프트웨어를 재사용(software reuse)할 수 있도록 해 주는 동시에, 재사용 가능한 소프트웨어(reusable software)를 생산할 수 있도록 함으로써 체계적인 재사용의 선순환 관계를 지원하게 된다.

2. 관련 연구

2.1 background

소프트웨어의 재사용성의 측면에서 도메인 엔지니어링과 소프트웨어 엔지니어링은 선순환적인 관계가 있다. 그림 1은 이러한 관계를 보여주고 있다.

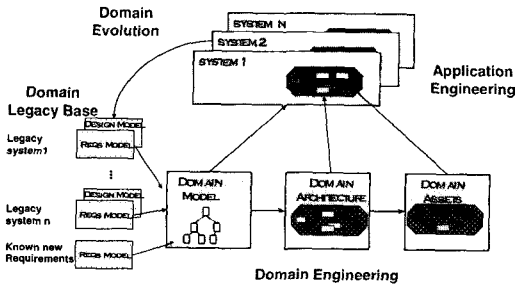

그림 1 도메인 엔지니어링의 선순환적 역할

즉, 기존의 개발되어진 소프트웨어 시스템-레거시(legacy) 시스템-에서 산출되어졌던 코드, 문서, 모델, 테스트 플랜과 같은 자산(asset)들을 재사용하고 잘 정의된 소프트웨어 개발 프로세스를 따르게 되면 차후 소프트웨어 개발에 있어서 품질과 신뢰성, 생산성의 향상을 가져올 수 있게 된다[2]. 소프트웨어 개발자들이 새로운 시스템을 개발하는 데 재사용 할 수 있는 부품들을 식별, 개발, 보급하는 것이 바로 도메인 엔지니어링(domain engineering)의 목적이다[3]. 도메인 엔지니어링에서는 도메인 분석, 아키텍처 개발과 재사용 가능한 자산 개발을 포함하며 이들은 모델, 아키텍처, 재사용 컴포넌트, 소프트웨어 생성기(software generator)를 생산한다.

어플리케이션 엔지니어링은 고객의 요구사항을 분석하고 도메인에서 이미 개발된 컴포넌트들을 조립하여 어플리케이션을 개발하는 활동이다. 이 과정에서 도메인 엔지니어링에서 산출되어진 자산들은 응용 소프트웨어 개발 각 단계에 적절히 사용되고 또한 다른 응용 소프트웨어 개발에 반복적으로 재사용 될 수 있는 프로덕트 라인으로써 활용된다.

2.2 도메인 분석 관련 기존 연구

2.2.1 Feature를 이용한 도메인 분석

1990년대 초 이후 시스템의 집합 중에서 주도적인 또는 독특한 피처(feature)를 인식하는 것에 기초해 도메인을 분석하는 방법들이 나왔다. 대표적인 방법으로는 Feature Oriented Domain Analysis(FODA)가 있다[4,5].

FODA에서 '피처(feature)'라는 것은 구현되고 테스트되고 배포, 유지되어져야 하는 기능적 추상화를 뜻하는 것으로 요구 사항이나 기능들을 의미한다. 이러한 피처들로 이루어진 피처 그래프는 도메인 내에서 공통성과 다양성을 추출하기 위한 도구로서 사용된다. 이 방법에서는 '추상화(abstraction)'와 '정제화(refinement)' 두 가지 모델링 개념에 기반을 두고 있다. 이는 어플리케이션들 사이에 어떠한 차이도 존재하지 않는 수준까지 '추상화'되어야 한다는 것이고, 도메인 상에 있어서 특정 어플리케이션은 도메인 프로덕트들의 '정제화'로서 개발되어진다는 것을 의미한다. FODA는 후에 소프트웨어 설계와 구현 단계 부분까지 확장되어 FORM(Feature Oriented Reuse Method) 방법론으로 이어진다.

FORM은 재사용 가능한 아키텍처와 컴포넌트 개발, 그리고 도메인 엔지니어링(domain engineering)에서 생성된 결과들을 이용한 어플리케이션 개발을 지원한다[6]. FORM은 도메인 엔지니어링(domain engineering)과 어플리케이션 엔지니어링(application engineering)으로 구성되며, 도메인 엔지니어링에서는 도메인 시스템들을 분석하고, 분석 결과에 기초해 참조 아키텍처(reference architecture)와 컴포넌트를 생성한다. 어플리케이션 엔지니어링에서는 도메인 엔지니어링에서 생성된 결과들을 사용하여 어플리케이션을 개발한다.

1990년대 후반 FODA의 피처 모델링 방법에 Reuse-Driven Software Engineering Business(RSEB)의 프로세스를 통합한 FeatuRSEB(the Featured RSEB) 방법이 나왔다[7]. 이 방법은 피처 모델을 만드는 활동과 병행하여 유즈케이스 모델을 만든다. 이는 도메인 내의 어플리케이션들의 일반적인 능력들을 캡쳐하기 위하여 객체 지향 요구사항 분석 방법인 유즈케이스 모델링 방법을 추가한 것이 특징이다. 이 유즈케이스 모델은 도메인 내의 각각의 유즈케이스 모델을 합하여(merge) 도메인 유즈케이스 모델을 만들어 낸다.

2000년도 이후에 소프트웨어 프로덕트 라인 개발에서 다양성을 표현하기 위한 연구에 피쳐가 사용되었다[8]. 여기서 사용되는 피쳐는 프로덕트들 사이에 나타날 수 있는 차이성을 기술하기 위한 용어로서 사용된다. 이는 소프트웨어 프로덕트 라인의 목적이 생산되는 시스템의 변경을 허용하는 것이기 때문에 아키텍처들 사이에 동일한 요소들보다 변할 수 있는 요소들을 인식하는 것에 중요성을 두고 있기 때문이다.

이처럼 도메인을 분석하는 분야에서 피쳐는 다방면에서 사용되어졌다. 그러나 피쳐를 분류하는 과정에 있어서 구체적인 방법을 제시하고 있는 것은 없었다. 피쳐의 분류는 모두 도메인 분석가의 경험이나 직관(heuristic)에 의존되어 있었고 그것은 문헌에서도 여러 군데 발견되었다[9].

2.2.2 Organization Domain Modeling(ODM)

ODM (Organization Domain Modeling)은 형식적이고 반복적인 도메인 엔지니어링 방법을 제공하며, 크게 Plan Domain단계, Model Domain단계와 Engineer Asset Base단계로 진행된다[1]. 이 방법은 프로젝트와 도메인에 대한 관심을 두고 있고, 다양한 관점과 경험을 가지고 있으며, 다양한 용어를 사용하는 stakeholder에 초점을 두었다. 또한 도메인 예(examples)의 명시적인 집합인 exemplar를 기반으로 모델링하였다. 각 단계는 하위 레벨 3까지 세분화시켜 전체 27개 단계를 제시하고 있으며, 각 단계에 대한 입력물, 출력물, 제어요소, 사용되는 툴들을 좌우상하 화살표로 표시한 컨텍스트 다이어그램(context diagram)으로 나타냄으로써 명확한 과정을 보여준다. 그러나 ODM은 도메인 아키텍처와 구현을 개발하기 위한 방법에 대해서는 구체적으로 규정하지 않았다. 또한 하위 레벨 도메인 객체를 분석하는데 중점을 잘못 두고 있다는 평가가 있기도 하다[7].

ODM을 기반으로 해서 도메인 아키텍처를 개발하고 개발된 도메인 자산들을 어플리케인션 개발에 적용시키는 단계까지 확장한 DAGAR 프로세스가 있다. DAGAR은 Ada에 기반을 두고 도메인 자산(asset)을 구현하기 때문에 도메인 아키텍처를 만들기 위한 2가지 기본 요소도 Ada의 용어인 '영역(realm)'과 '컴포넌트(component)'로 제시한다[3]. 영역(realm)은 도메인 자산에 의해 제공되어지는 핵심 서비스를 계층화된 도메인 아키텍처 내에서 나타내는 것이다. 그러나 DAGAR은 하나의 특정 프로그램 언어 Ada를 기반으로 하고 있어 그 사용에 한계가 있다.

2.3 컴포넌트 기반 소프트웨어 개발 방법

CBSD(Component-Based Software Development)는 이미 존재하는 소프트웨어 컴포넌트를 조립함으로써 시스템을 개발하는 방법이다[10]. 이와 관련된 프로세스 방법론들은 학계와 산업계에서 모두 활발하게 진행되었다.

Catalysis는 UML을 기반으로 한 컴포넌트 기반 개발 방법을 지원한다[11]. 이 방법은 어떤 프로젝트에서

도 적합하게 사용될 수 있도록 프로세스 패턴(process pattern)을 제공한다. Catalysis는 비즈니스 목적부터 프로그램 코드까지 이르는 컴포넌트 모델링 방법과 컴포넌트의 명확한 명세 기법을 제시하고 있다. 그러나 구체적으로 컴포넌트를 추출하고, 아키텍처를 구성하는 방법을 제시하기에는 미흡하다. 또한 Catalysis는 컴포넌트에 대한 가장 흥미있는 이론적인 접근 방법이나 사용하기에 너무 학술적이고 복잡하다[12].

스터링 소프트웨어(Sterling Software)의 Advisor 방법론은 Catalysis 방법을 사용하기 쉽도록 단순화하고 체계화하고, 확장했다[12,13]. 스터링의 CBSD 프로세스는 Requirements Definition, Analysis, Behavior Specification, Architecture 4단계로 이루어진다. Sterling에서는 여러 가지 case tool을 개발하여 CBSD 프로세스를 지원한다. 그러나 각 단계에서 생성되는 모델의 표현에 중점을 두고, 그 모델의 구체적인 생성방법을 제시하지 않았다.

Unified software development process는 UML을 사용하여 컴포넌트 기반 소프트웨어를 개발하는 방법론으로서 Inception 단계, Elaboration 단계, Construction 단계, Transition 단계의 크게 네 단계로 구성되어 있고 각 단계 내에는 Requirements, Analysis, Design, Implementation, Test의 다섯 개의 Core Workflows로 구성된다[14]. 각 Core Workflows를 수행하면서 UML diagram과 element로 구성된 모델을 생성한다. 그러나 Unified process의 한계는 다섯 개의 core workflow를 거치면서 수행되는 여러 작업들을 통해 생성되는 각 모델을 구성하는 구성요소들에 대하여 정의는 하고 있지만 각각의 작업간의 관계와 각각의 작업에서 입력으로 제공되는 정보가 각 활동에 어떤 영향을 주고 그런 활동의 결과로 각 모델의 구성요소들이 어떻게 생성되는지에 대하여 체계적으로 제시하고 있지 않다.

UML Components는 Syntropy, OMT, Catalysis, Advisor 등 여러 방법론에 영향을 받아 개념 모델을 정련하고 RUP(Rational Unified Process)의 용어와 워크플로우를 사용하여 개발 프로세스를 제시한다. 관리 프로세스와 분리되어 있으며 특히 컴포넌트 시스템의 아키텍처와 의존성을 명세화하는 방법을 집중적으로 다루고 있다. 매우 실용적이며 특정 회사의 제품에 종속적인 경향이 적으며 표준 UML을 사용하여 컴포넌트를 모델링 할 수 있는 기법을 제시하고 있다[15].

국내 기술진에 의해서 자체 개발된 마르미-III(Magic and Robust Methodology Integrated)은 컴포넌트 개발 및 컴포넌트 기반의 시스템 개발에 필요한 작업과 작업 수행에 필요한 기법 및 각 작업별 산출물을 정의하고, 작업에 따른 상세한 개발 절차와 지침을 제공한다. 개발 절차는 크게 계획 단계, 아키텍처 단계, 점진적 개발 단계, 인도 단계로 이루어져 있으며, 작업공정을 몇 개의 층으로 나누어 번호를 통해 프로젝트를 관리할 수 있는 체계를 만들었다[16]. 마르미-III는 여러 개의 미니프로젝트를 통하여 반복적이고 점진적인 개발을 지원하지만, 실질적으로 일부 단계에서만 반복 개발의 개념이 적용되고 있어 포괄적인 지원은 못하는 것처럼 보인다.

이들 방법론에서 각기 상이한 면을 발견할 수 있지만 기본적인 컴포넌트 기반의 설계에 있어서는 근본적으로 동일점이 많다. 이들 방법론은 CBD를 구성하는 핵심 요소로 컴포넌트, 인터페이스, 그리고 컴포넌트의 조립을 공통적으로 말하고 있다.

3. 도메인 설계 방법

본 논문에서 제시하는 컴포넌트 기반 도메인 엔지니어링의 전체 상호 과정은 그림 2와 같다.

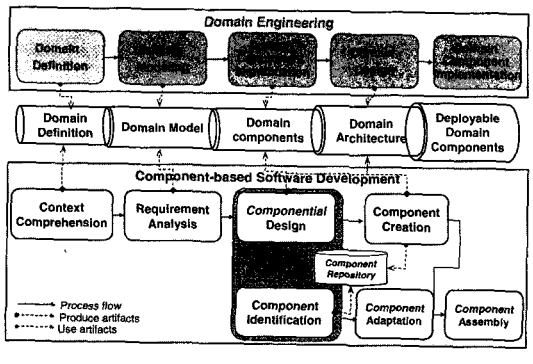

그림 2 도메인 중심의 컴포넌트 기반 소프트웨어 개발 프로세스

컴포넌트 기반 소프트웨어 개발 프로세스는 먼저, 개발할 어플리케이션이 놓일 문맥을 이해를 하고 요구사항을 분석하는 과정을 거친다. 이를 통해 요구사항에 맞는 컴포넌트를 인식하고 어플리케이션의 특성에 맞게 컴포넌트를 적응시킨다. 이 과정에서 인식되어진 컴포넌트가 존재하지 않을 경우에는 컴포넌트를 생성한다. 마지막으로 컴포넌트를 조립하여 어플리케이션을 만들게 된다.

컴포넌트 기반 도메인 엔지니어링의 목적은 컴포넌트

기반 소프트웨어 개발을 지원하는 것이므로 프로세스가 이에 의존적이게 된다. 도메인 엔지니어링 프로세스는 먼저, 도메인 정의 단계에서는 도메인의 목적을 정하고 도메인의 범위를 확정한다. 도메인 모델링 단계에서는 도메인 내에 존재하는 여러 어플리케이션의 요구사항들을 추출하여 모델링 시키게 된다. 이때 요구사항들의 공통성과 다양성을 분리하여 구분시킨다. 이를 바탕으로 도메인 컴포넌트를 인식하게 되고 도메인 아키텍쳐를 만들게 된다. 이 과정에서 생성되는 산출물들은 서로의 연관성을 유지하며 저장된다. 이는 컴포넌트 기반 소프트웨어 개발 시 유용한 정보로써 재사용 된다.

본 논문에서는 그림 2에서 제시한 도메인 중심의 컴포넌트 기반 소프트웨어 프로세스 전체 연구 중 도메인 엔지니어링 과정의 도메인 정의, 분석, 컴포넌트 인식, 설계 단계에 대하여 우선 제시하고자 한다. 이 과정에 대한 세부 활동들은 그림 3과 같다.

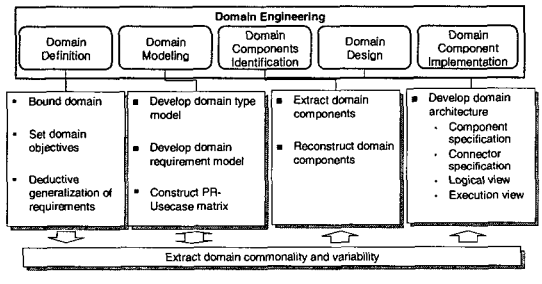

그림 3 도메인 엔지니어링 세부 활동들

3.1 도메인 정의

도메인 정의 단계에서는 개발하고자 하는 도메인의 범위를 한정하고 목적을 정의하여 도메인 명세서를 생성한다. 도메인 명세서는 도메인에 대한 전반적인 정보를 제공하는 문서로, 도메인 요구사항을 도출하고, 아키텍처를 개발하기 위해 사용된다.

3.1.1 도메인 범위 결정

앞에서 정의한 바처럼 도메인은 일련의 관련된 시스템의 집합이다. 이는 매우 모호한 의미를 띌 수 있기 때문에 그 범위를 명확히 하는 것이 매우 중요하다. 본 논문에서는 이러한 도메인의 범위들 사이에 일반화의 개념을 적용하여 범위가 더 넓은 도메인을 상위 도메인이라 하고, 그로부터 제약이 더 가해지는 범위가 좁은 도메인을 하위 도메인이라 한다. 즉, 하위 도메인은 상위 도메인이 가지는 공통성의 요소를 가지면서 나름대로의 공통성의 요소를 더 가질 수 있게 된다. 이러한 관계사이의 트레이드 오프를 고려하여 도메인의 범위를 결정하는 것은 다음 단계의 도메인 분석에 많은 영향을 끼치게 된다.

① 도메인 외부 stakeholder를 식별한다.

도메인 외부 stakeholder는 도메인에서 제공하는 기능들에 관심이 있는 사람들로 도메인의 입력, 출력에 관심이 있는 사람, 도메인과 관련된 외부 시스템을 다루는 사람이 추출된다.

② 도메인 가정을 정한다.

도메인에서 제공하는 컴포넌트를 사용하기 위해 만족해야 할 전제 조건들이 도메인 가정이 될 수 있다. 도메인 가정은 초기 단계에는 시스템의 포함 여부를 결정하는 기준 역할을 하고, 후에는 추출되는 도메인 컴포넌트의 속성을 결정하는데 영향을 주게된다.

③ 도메인 환경을 기술한다.

도메인 환경은 도메인 외부 환경(domain external environment)과 도메인 내부 환경(domain internal environment)으로 나눈다. 도메인 외부 환경은 도메인과 도메인 외부 요소들과의 상호작용을 나타냄으로써 도메인의 경계를 명확하게 표현한다. 도메인 내부 환경은 도메인 내에서 반드시 분산되어야 하는 요소들과 그 요소가 제공하는 기능을 표현한다.

3.1.2 도메인 목적 정의

도메인의 범위가 정해지면 그 도메인의 기능적인 측면을 중심으로 대략적인 윤곽을 설명한다. 또한 도메인 목적과 관련하여 도메인 내의 비즈니스 프로세스를 이해하기 위해 도메인 개념 다이어그램을 그린다.

① 도메인 목적을 기술한다.

도메인이 가지는 중요한 기능들을 기술한다. 이는 그 도메인에 속한 모든 시스템이 반드시 가져야 하는 요소로서 도메인 내에 해당 시스템의 포함 여부를 결정지을 수 있는 기준으로 역할을 하게 된다.

② 도메인 개념을 모델링한다.

도메인 내에서 명확히 정의해야 할 필요가 있는 주요 업무와 이와 연관된 중요한 용어들을 추출하고, 이들 간의 연관 관계를 식별하여 전체적인 그림으로 도식화시킴으로써 도메인 내의 개념들을 표현한다. 도메인 개념 모델은 UML의 클래스 다이어그램의 형태로 표현한다.

3.1.3 연역적 방법을 통한 도메인 요구사항 일반화 과정

대부분의 경우 도메인 범위 내에 이미 개발된 시스템들이 존재하게 된다. 이러한 경우 기존에 있는 레거시 시스템들로부터 요구사항들을 추출하여야 하며, 또한 이를 재사용 할 수 있는 형태로 일반화시킬 필요가 있다.

일반화된 도메인 요구사항들은 어플리케이션 개발 시뿐만 아니라 다음 번 도메인 재분석 시, 하위 도메인의 요구사항 분석 시에도 재사용될 수 있다.

· 도메인 요구사항 추출 과정이 일반 어플리케이션 요구사항 추출과 가장 두드러지는 차이는 요구사항들에서 도메인 내의 공통된 요구사항과 선택적 요구사항, 다양성을 가진 요구사항들로 그 속성이 구분 지어져야 한다는 것이다. 이러한 요구사항들의 속성을 구분 짓는 활동을 '도메인 요구사항 일반화 과정'이라 정의한다. 이 과정에서 구분된 요구사항 속성들은 그 정보를 계속 유지, 정제시켜 나가며 도메인 분석, 도메인 컴포넌트 추출 시에도 활용하게 된다. 분석된 요구사항들은 지금까지 도메인 전문가의 경험과 본능적인 감각(heuristic)에만 의존해서 인식하게 되는 공통성과 다양성의 추출에 좀 더 객관화 된 정보로서 사용할 수 있게 된다.

① 도메인 요구사항을 수집한다.

도메인내의 시나리오들을 분석하여 atomic 레벨 컴포넌트의 의미상의 최소 단위인 semantic primitive[17]로 나눈다. 나누어진 semantic primitive를 여기서는 primitive requirement(PR)라고 부른다. 요구사항 수집의 초기 과정에는 새로운 PR을 연속적으로 만들어 가며 요구사항들을 추출한다. 도메인 내의 레거시 시스템들에 대한 요구사항들을 반복적으로 분석하는 과정에서 동일한 PR이 계속 나오게 된다.

② PR-Context 메트릭스(domain Primitive Requirement-context matrix)를 구축한다.

도메인 안에 존재하는 일련의 시스템들에 대해 앞 단계에서 수집한 domain primitive requirement의 존재 유무를 그림 4에서 나타낸 바와 같이 메트릭스 형태로 표현한다. 이 과정에서 임의의 어플리케이션 시스템들이 동일한 primitive requirement들로 분해되는 경우, 이들을 context라는 용어를 사용하여 하나로 묶고 동일하게 취급될 수 있도록 한다. 즉, 동일한 요구사항들로 이루어진 시스템들은 같은 문맥을 가지고 있다고 볼 수 있기 때문에 하나로 그룹시킬 필요가 있다. 도메인 내의 레거시 시스템들에 대해 요구사항들은 분해시키고 시스템들은 그 요구사항들의 분석을 통해 다시 합치는 과정을 반복적으로 거치면서 PR-Context 메트릭스가 완성된다.

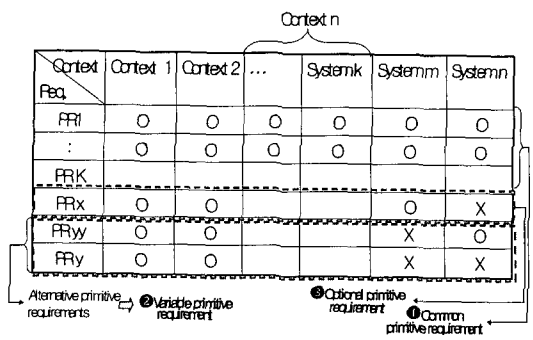

그림 4 PR-Context 메트릭스

③ domain primitive requirement들을 일반화시킨다.

이전 단계에서 만들어진 PR-Context 메트릭스를 분석하여 공통성, 선택성, 다양성을 지닌 요구사항들로 분류를 한다. 그림 4의 ①처럼 대다수의 context에서 필요로 하는 primitive requirement인 경우에 이를 해당 도메인 내에서 공통성을 가진 요구사항으로 인정을 한다. 여러 context에 선택적으로 나타나는 primitive requirement인 경우에는 그와 대치할 수 있는 primitive requirement가 존재하는 지를 확인한다. 그림 4의 ②와 같이, 만약 서로 비슷한 성질을 가진 요구사항들이 발견될 시에는 이를 그룹지어 다양성을 가진 요구사항으로 인정을 한다. 다양성의 속성은 좀 더 세분화하여 공통성 요구사항에서 나타나는 다양성과 선택성 요구사항에서 나타나는 다양성으로 나뉠 수 있다. 또한 그림 4의 ③처럼 다른 요구사항과는 별개의 성질을 가지고 여러 context에서 선택적으로 나타나는 requirement인 경우에는 이를 선택적 요구사항으로 인정한다.

3.2 도메인 모델링

도메인 모델링은 도메인 정의 단계에서 추출된 도메인 명세서를 바탕으로 도메인의 정적인 특징들과 동적인 특징들을 도식화하여 도메인을 분석한다. 그 결과 생성되는 산출물은 도메인 요구사항 모델과 도메인 타입 모델로 구성된다. 또한 산출되는 모델에 공통성과 다양성의 정보를 제공하기 위하여 PR-Context 메트릭스를 참조하여 Primitive Requirement-Usecase 메트릭스(PR-Usecase matrix)를 생성한다. 도메인 모델과 메트릭스의 완성 과정은 서로 정보를 주고받으며 병행적으로 수행되며 완성된다.

3.2.1 도메인 요구사항 모델 개발

도메인 요구사항 모델은 도메인에서 추출된 요구사항들을 UML의 유즈케이스 다이어그램을 이용하여 표현한다. 이는 이전 단계에서 추출된 primitive requirement가 다시 적정한 단위로 묶어지는 결과를 유도한다. 묶어진 유즈케이스 단위는 컴포넌트를 추출해 내는 기초 정보로 사용된다. 도메인 요구사항 모델은 도메인 유즈케이스 모델과 도메인 유즈케이스 서술서(domain

usecase description)로 구성된다.

① 도메인 유즈케이스 모델을 그린다.

도메인 stakeholder와 도메인 외부 환경으로부터 엑터를 추출한다. 이러한 엑터가 가지는 요구사항, 그리고 도메인 오퍼레이션을 요구사항으로 추출한다. 마지막으로 엑터에게 결과를 제공하는 단위로 요구사항을 재정의하여 유즈케이스 다이어그램을 작성한다. 유즈케이스는 여러 번의 반복 과정을 통하여 여러 레벨의 유즈케이스로 정제될 수 있다.

② 도메인 유즈케이스 서술서를 기술한다.

도메인 유즈케이스 서술서는 도메인 유즈케이스 모델에서 정의한 각 유즈케이스에 대한 정보를 제공한다. 본 논문에서는 각 도메인 유즈케이스에 대해 텍스트로 그림 5와 같은 항목을 기술한다.

Use case Name
Goal
Precondition
Actors
Activation condition
flow of events
main flow
- describe what system does and what the actor does
- how to start use case
- system interaction with the actors and what they exchange
- paths of execution that are not allowed
- usage of objects, values, and resources in the system
- how and when the use case ends
*- reference to **[optional part1 / optional part2]***
*- reference to **@variable part***
alternative flow
Postcondition
successive postcondition
failed postcondition
Variation

그림 5 도메인 유즈케이스 서술서

이때, PR-Context 메트릭스를 참조하여 분리되어 나온 다양성 요구사항의 참조 지점은 템플릿 마켓(template market)-⊕을 달아 표현한다. 마찬가지로 선택적 요구사항의 언급 시에는 선택 심볼 '[', '/', ']'를 사용하여 표현한다.

3.2.2 도메인 타입 모델 개발

도메인 타입 모델은 도메인 정의 단계에서 만들어진 도메인 개념 모델과 바로 이전 단계에서 나온 도메인 유즈케이스 서술서를 바탕으로, 시스템에 의해서 관리되어야 할 구체적인 정보, 즉 유지 관리할 필요가 있는 데이타나 상태를 추출하여 나타낸다. 여기서는 물리적인 것뿐만 아니라 프로세스와 같이 물리적으로 나타나지 않는 것들도 도메인 타입이 될 수 있다. 이는 도메인에 대한 공통적 이해와 프로세스 전체에 일관된 용어(glossary) 적용을 가능토록 하는 역할을 한다.

도메인 타입 모델은 UML의 클래스 다이어그램의 형태로 표현한다. 각 도메인 타입 별로 속성을 정의하며, 연관관계의 다중성(multiplicity)과 같은 모델의 제약 조건 등을 정의한다.

3.2.3 PR-Usecase 메트릭스 (domain Primitive Requirement - Usecase matrix) 구축

유즈케이스 다이어그램이 도식화되면, 도메인 정의 단계에서 생성된 PR-Context 메트릭스를 참조하여 PR-Usecase 메트릭스를 만든다. 구축 과정은 유즈케이스 서술서를 바탕으로 각각의 유즈케이스가 어떠한 primitive requirement로 이루어져있는가를 분석하여 메트릭스 형태로 표현한다. 메트릭스에는 유즈케이스 이름, primitive requirement 뿐만 아니라 이전 단계에서 추출된 primitive requirement에 대한 속성들이 표시된다. 다음 그림 6은 PR-Usecase 메트릭스의 구축 형태를 보여준다.

Usecase / Req.	property	Usecase1	Usecase2	Usecase3	Usecase4		Usecase k
PR1	c	O					
PR 2	c	O					
PR k	:	❶	O				
PR m			O	O			❸
PR n			❷	O			O
:					O		
PR k	v				O		
PR m	p				❹	O	O

c: common PR v: variable PR p: optional PR
cv: common PR with variables pv: optional PR with variables

그림 6 PR-Usecase 메트릭스

이는 유즈케이스가 어떠한 속성의 요구사항을 가지고 있는가를 분석할 수 있게 해 주며 이를 바탕으로 유즈케이스를 재정비하게 된다. 이는 도메인 정의 단계에서 추출된 공통성과 다양성의 속성이 그대로 도메인 모델에 반영시키기 위한 작업이며, 이를 바탕으로 이후에 도메인 컴포넌트의 속성을 판단할 수 있게 해 준다.

유즈케이스를 분석해 보면 다음과 같이 나누어 고려해 볼 수 있으며 이때 필요에 따라 유즈케이스를 분리시켜 재정비시킬 수도 있게 된다. 이는 그림 6과 그림 7에서 보여준다. 고려될 수 있는 유즈케이스의 상황은 첫째, 유즈케이스가 다른 유즈케이스에 겹치지 않는 primitive requirement들로만 이루어진 경우이다(그림 6과 7의 ① 경우). 이때는 아무런 재정비도 필요하지 않

다. 둘째는 유즈케이스에 포함된 primitive requirement가 여러 유즈케이스에 걸쳐 있는 경우이다(그림 6과 7의 ② 경우). 이 때는 공통적으로 겹치는 부분을 따로 분리하여 독립된 유즈케이스로 만든 후, 이를 include 관계로 연관시킨다. 셋째는 유즈케이스가 다양성 primitive requirement를 포함하고 있는 경우이다(그림 6과 7의 ③ 경우). 이 때는 다양성 부분이 분리되어 독립된 형태로 존재 가능한지를 확인하여 가능한 경우에는 이를 분리시키고 그렇지 못한 경우에는 그대로 내포한 상태를 유지하며 그 정보를 가진다. 넷째는 유즈케이스가 선택적 primitive requirement를 포함하고 있는 경우이며(그림 6과 7의 ④ 경우), 이 때는 선택적 primitive requirement들을 분리하여 extends 관계로 연관시킨다.

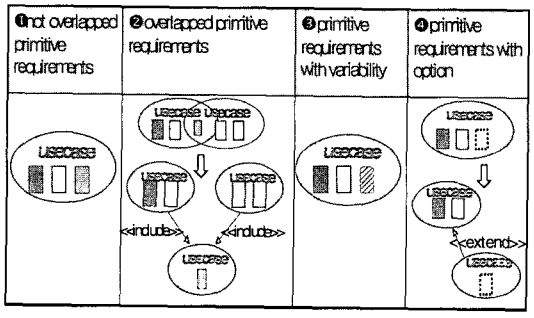

그림 7 PR-Usecase 메트릭스로부터 속성 구별 과정

이 과정을 거쳐 재정비된 유즈케이스의 속성은 다음과 같이 분류한다.

① 공통성 유즈케이스(common use-case)

유즈케이스가 그 도메인 내에 반드시 있어야 하는 요구사항들을 가지고 있는 경우, 공통성 유즈케이스로 분류되며 이는 중요한 프로세스를 나타내게 된다.

② 다양성 유즈케이스(variable use-case)

유즈케이스 그 자체의 요구사항들이 다양성의 속성을 가지고 있는 경우, 이는 다양성 유즈케이스(variable use-case)로 구분된다. 즉, 다양성 유즈케이스는 도메인 내의 특정 어플리케이션마다 반드시 있어야 하지만 달라질 수 있는 요구사항들을 가진 유즈케이스를 의미한다. 주로 여러 유즈케이스에 겹쳐 나타났지만 PR-Usecase 메트릭스 분석을 통해 다양성을 지닌 독립된 유즈케이스로 분리된 경우에 이 부류에 속하게 된다.

③ 선택적 유즈케이스(optional use-case)

시스템의 프로세스를 처리하는데, 반드시 존재하지 않아도 가능한 유즈케이스를 나타낸다. 이는 선택적 요구사항으로 구성된 유즈케이스인 경우에 해당한다.

④ 다양성을 가진 유즈케이스 (usecase with variables)

다양성의 속성을 가진 primitive requirement가 독립적으로 분리되기 어려운 경우 이는 유즈케이스 내부에 포함된다. 이는 따로 분리 할 수는 없지만 그 상태를 구분 시켜 줌으로써 다음 단계의 도메인 컴포넌트 추출과 도메인 설계 단계에서 컴포넌트 상호 작용을 도식화 할 때 이용할 수 있다.

3.3 도메인 컴포넌트 인식

컴포넌트 기반 소프트웨어 개발에서 가장 중요한 과정은 컴포넌트를 추출하는 활동이다. 그러므로 컴포넌트 기반 도메인 분석 시에도 이 과정이 역시 중요한 역할을 하게 된다. 특히 도메인 컴포넌트의 추출은 도메인 모델에서 표현되고 있는 공통성과 다양성, 선택적 속성들을 반영하고 있도록 컴포넌트가 추출되어져야 한다.

본 논문에서는 도메인 컴포넌트 추출이 유즈케이스를 기반으로 이루어지기 때문에 이전 단계에서 만들어진 유즈케이스 모델의 레벨과 속성을 그대로 이용할 것이다. 추출된 컴포넌트는 그 속성을 분석하여 재구축 과정을 거쳐 최종 도메인 컴포넌트로 인식해 낸다.

3.3.1 도메인 컴포넌트 추출

도메인 컴포넌트는 소프트웨어 개발 시 바로 배치(deploy)될 수 있는 물리적 컴포넌트와는 달리, 플랫폼 독립적인 논리적 수준의 서비스 중심 단위 패키지로 정의한다.

도메인 컴포넌트는 도메인 모델에서 구분시켜놓은 유즈케이스를 바탕으로 분석을 하고, 유즈케이스가 가지는 속성들에 대한 정보를 이용하여 공통성을 가진 컴포넌트, 다양성을 가진 컴포넌트, 선택적 컴포넌트로 분류하여 추출한다. 컴포넌트를 추출하는 과정은 다음과 같다.

① 각 유즈케이스에 대하여 엑터를 분류한다.

- 유즈케이스의 수행을 시작하도록 하는 엑터(Initiate actor) : 유즈케이스에 해당되는 기능을 수행하는 처음 단계의 입력을 제공하는 입력 제공자 엑터가 해당된다.
- 유즈케이스의 수행에 대한 최종 결과를 사용하는 엑터(UseOutput actor) : 유즈케이스에 해당되는 기능을 수행하는 마지막 단계의 출력을 제공받는 출력 리시버 엑터가 해당된다.
- 유즈케이스의 기능 수행에 참여하는 엑터(Participate actor) : 기능의 수행 결과를 생성하기 위한 과정 중에 참여하는 입력 제공자, 출력 리시버, 리소스 엑터가 해당된다.

② 도메인 컴포넌트를 추출한다.

도메인 컴포넌트는 유즈케이스와 액터를 바탕으로 서비스 제공과 상호 독립성을 만족시키는 기능 중심의 독립적인 단위로 추출한다. 도메인 컴포넌트 추출 기준은 다음과 같이 5가지로 분류한다.

• 기본적으로 서비스의 단위가 될 수 있는 것은 서비스에 해당되는 기능을 수행함으로써 특정 액터가 원하는 결과를 얻고자 하는 것이다. 그림 8의 (a)에서처럼 유즈케이스 모델에서 하나의 usecase1에 대해 수행 결과를 사용하고자 하는 Actor1이 있는 경우 Actor1이 원하는 수행 결과를 얻기 위해서 필요한 기능이 도메인 컴포넌트로 추출된다.

• 특정 엑터에게 서비스를 제공하기 위해서 여러 유즈케이스가 관련이 있을 때 관련 있는 유즈케이스가 모두 요구된다. 그림 8의 (b)처럼 관계가 있는 유즈케이스들(usecase1, usecase2, usecase3)의 기능 수행 결과를 사용하고자 하는 Actor1이 존재하는 유즈케이스가 도메인 컴포넌트로 추출된다.

• 각 유즈케이스에 해당되는 기능을 수행하는 과정상에 참여하는 엑터가 기능을 수행해야만 하는 경우에 그러한 엑터의 기능도 유즈케이스 측면에서 하나의 서비스라고 할 수 있으며, 이를 하나의 도메인 컴포넌트로 추출한다. (그림 8의 (c))

• 각 유즈케이스에 대해 수행을 시작하게 하는 엑터가 존재하는 경우 그러한 엑터의 역할이 하나의 도메인 컴포넌트로 추출된다. (그림 8의 (d))

• 각 유즈케이스에 대해 수행 결과를 사용하는 엑터가 존재하는 경우 그러한 엑터의 역할이 하나의 도메인 컴포넌트로 추출된다. (그림 8의 (e))

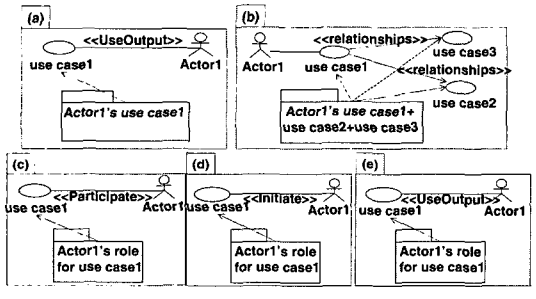

그림 8 도메인 컴포넌트 추출 기준

3.3.2 도메인 컴포넌트 재구성

추출된 각각의 도메인 컴포넌트는 유즈케이스를 바탕으로 관계를 형성하고 PR-Usecase 메트릭스를 참조하여 공통성을 가진 컴포넌트, 선택적 컴포넌트, 다양성을 가진 컴포넌트로 그 속성을 부여한다. 이 과정에서 필요에 따라 컴포넌트를 재구성하게 된다.

① 도메인 컴포넌트 간의 중복되는 유즈케이스는 별개의 단위로 추출하고 의존관계를 설정한다.

② 유즈케이스의 속성이 순수한 경우 이를 도메인 컴포넌트에 그대로 반영시킨다. 순수한 속성이란 유즈케이스가 가지고 있는 속성이 공통성, 다양성, 선택성의 한 가지 속성만으로 이루어진 경우를 의미한다.

③ 유즈케이스의 속성이 혼합되어 있는 경우 즉, 다양성을 내포하고 있는 경우에는 더 일반적인 형태로 변형한다. 즉, 도메인 컴포넌트 내의 다양성 부분을 구분하고 그 부분을 <<Hole>>로 대치함으로써 추상화시킨다. <<Hole>>은 어플리케이션 컴포넌트 생성 시마다 그 내용을 달리 채워 넣을 수 있는 부분이 된다.

3.4 도메인 설계

도메인 설계에서는 추출된 도메인 컴포넌트들 간의 관계를 인식하고 표현하여 도메인 아키텍처를 생성한다. 도메인 아키텍처에서는 논리적인 측면에서의 컴포넌트 간의 상호 작용 모습과 전체적인 컴포넌트와 컨넥트의 모습들을 표현하며 도메인 컴포넌트에 대한 명세와 컨넥터에 대한 명세를 보여준다. 도메인 아키텍처의 모습에는 요구사항 분석 단계로부터 생성되어 각 단계를 거치면서 정제, 유지되어 온 공통성, 다양성, 선택적 속성을 반영하여야 한다. 이러한 모습은 컴포넌트 기반 소프트웨어 개발 시 도메인 아키텍처에 표현되어 있는 컴포넌트 속성에 따라 아키텍처의 일부분이 분리, 대치될 수 있도록 해 주기 때문에 유연한(malleable) 아키텍처를 생성토록 한다. 본 논문에서는 이러한 정보들을 여러 가지 뷰로 분리하여 표현한다.

3.4.1 실행 뷰

실행 뷰는 특정 요구사항을 수행하기 위해 발생하는 컴포넌트간의 상호작용을 표현한다. 즉, 두 컴포넌트간의 실행 시 관계를 나타냄으로써 커넥터에 대한 정보를 제공하고 이를 바탕으로 컴포넌트 인터페이스를 추출하게 된다.

실행 뷰는 sequence 다이어그램과 collaboration 다이어그램을 사용하여 표현하며 컴포넌트간의 상호작용은 컴포넌트간에 교환되는 메시지를 통해 표현된다. 그림 9는 실행 뷰를 나타낸다.

실행 뷰에 나타나는 컴포넌트는 그 속성 그대로 구별하여 도식화 시켜준다. 실행 뷰에서 선택적 성질을 가진

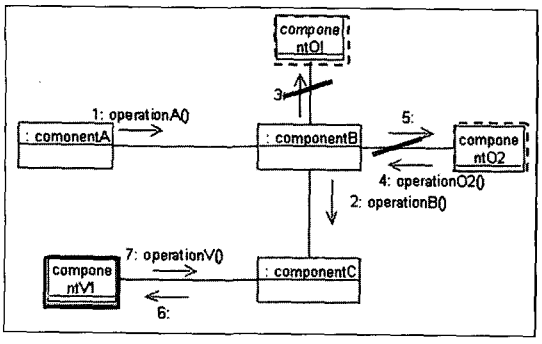

그림 9 컴포넌트 속성을 표현하고 있는 실행뷰

컴포넌트는 필요에 따라 가지가 쳐질 수 있다(prune). 이러한 과정을 거쳐 나온 실행 뷰는 공통성 요소만 가지고 있는 분석 모델이 된다. 이를 여기서는 Commonality Analysis view라 정의한다. Commonality Analysis view는 컴포넌트 기반 소프트웨어 개발 시 가장 기본 형태의 실행 뷰의 모습을 보여주게 되고 도메인 내의 특정 소프트웨어에 맞는 아키텍처 개발 시 변형이 용이하여 유연한 아키텍처의 역할을 한다.

3.4.2 논리 뷰

논리 뷰는 도메인 전체에 대해 컴포넌트와 커넥터, 그들간의 관계를 class diagram을 사용하여 표현한다. 논리 뷰는 요구사항마다, 또는 컴포넌트 단위마다 생성된 다른 뷰들을 하나로 통합시켜주는 역할을 한다.

논리 뷰의 컴포넌트는 UML의 패키지(package) 표기법을 사용하여 표현하며 패키지에는 <<component>>라는 스테레오타입을 기술한다. 그림 10은 논리 뷰를 보여준다.

다른 컴포넌트에게 서비스를 요청하는 required interface는 서비스를 제공하는 provided interface와 연결된다. 두 컴포넌트 인터페이스의 연결관계가 일반적인 function call이 아닌 경우, 커넥터를 사용하여 둘 사이의 연결관계를 명확하게 표현한다. 커넥트는 실행 뷰로부터 추출한다. 실행 뷰에서 제공하는 각 컴포넌트간에 실행 시 발생하는 관계를 기준으로 어떤 특성을 가지는 커넥터가 필요한 지 인식한다. 커넥터는 인터페이스 표기법을 사용하여, 속성 부분에는 커넥터가 연결시키는 컴포넌트 이름을, 동작 부분에는 커넥터의 연결 방법을 기술한다. 커넥터에 대한 좀 더 정확한 정보는 커넥터 명세에서 기술된다.

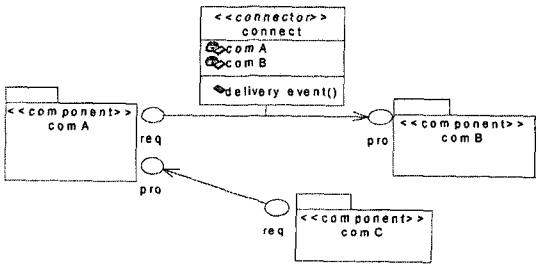

그림 10 package diagram 사용하여 표현한 논리뷰

3.4.3 컴포넌트 명세

컴포넌트 명세는 각 컴포넌트가 무엇을 하는지, 그리고 그 컴포넌트를 어떻게 사용할 수 있는 지에 대한 정확한 정보를 제공한다. 컴포넌트는 컴포넌트의 목적, 컴포넌트 구성 인터페이스, 포트(port), 동일하거나 유사한 기능을 제공하는 컴포넌트 항목으로 기술된다. 포트는 컴포넌트가 가지는 인터페이스 중에서 외부에서 볼 수 있는 오퍼레이션이다. 포트에서는 컴포넌트의 인터페이스에 정의된 각 오퍼레이션에 관한 정보를 제공한다.

3.4.4. 커넥트 명세

커넥트는 컴포넌트들 사이의 상호작용과 협력에 대하여 설명하는 개념적인 단위이다. 커넥트는 커넥터 타입, 역할(Role), 상호 프로토콜 항목으로 기술된다. 커넥트 타입은 이벤트, 메시지, 큐와 같은 커넥터의 종류를 말한다. 역할은 커넥터가 연결시켜주는 컴포넌트의 협력 책임(collaboration responsibility)을 기술한다. 예를 들어, 클라이언트-서버 커넥트는 클라이언트 역할과 서버 역할을 가지고, 이벤트-통지(event notification) 커넥트는 "EventPushSupplier", "EventChannel", "Event PushConsumer"와 같은 다양한 역할을 가진다. 상호 프로토콜 항목에서는 다양한 역할들의 행동이 어떻게 조화되어 움직이는 지를 순서적으로 기술하는 부분이다. 이는 UML의 순차(sequence) 다이어그램을 이용하여 나타낸다.

3.5 도메인 산출물간의 관계

도메인 엔지니어링의 각 단계별 산출물 관계를 그림 11에서 보여준다. 도메인 정의 단계에서 도메인 범위를 한정하고 도메인 목적을 기술하고 도메인 안에 있는 개념들에 대해 다이어그램으로 그려 놓은 도메인 개념 모델이 도메인 명세서에 서술된다. 도메인 모델링 단계에서는 유즈케이스 모델과 유즈케이스 서술서, 도메인 타입 모델이 만들어진다. 이 두 과정에서는 도메인 내에 있는 공통성과 다양성을 추출하기 위해 각각 메트릭스를 만들게 된다. 도메인 정의 단계에서는 PR-Context 메트릭스를 만들고 도메인 모델링 단계에서는 PR-

Usecase 메트릭스를 만든다. 이 메트릭스들은 그 정보를 계속 정제시켜가며 각 단계의 산출물에 공통성과 다양성에 대한 속성을 객관적으로 부여하는 역할을 하게 된다. 도메인 컴포넌트 인식 단계에서는 도메인 안에 존재하는 기능들을 중심으로 컴포넌트를 추출하게 된다. 도메인 설계 단계에서는 추출된 도메인 컴포넌트를 가지고 서로 상호 작용하는 모습들을 여러 가지 뷰를 통해 보여주는 도메인 아키텍처를 산출하게 된다.

4. 사례 연구

본 논문에서는 두께가 6 mm 이상의 강판이며, 선박 건조와 교량 및 각종 산업기계 등에 널리 쓰이는 후판의 입고, 출고, 이적 업무를 자동으로 수행하는 후판 창고관리 시스템 도메인에 제안한 도메인 분석, 설계방법을 적용하였다.

4.1 후판 창고관리 시스템 도메인 명세서

기존에 개발된 다수의 후판 창고관리 시스템들과 그로부터 추출된 요구사항들을 입력으로 하여 도메인 범위를 한정하고 도메인 목적을 기술하며 도메인 개념 모델을 그려 도메인 명세서를 작성한다. 그림 12는 도메인 목적을 정의하는 단계에서 산출되는 도메인 개념 모델을 보여준다.

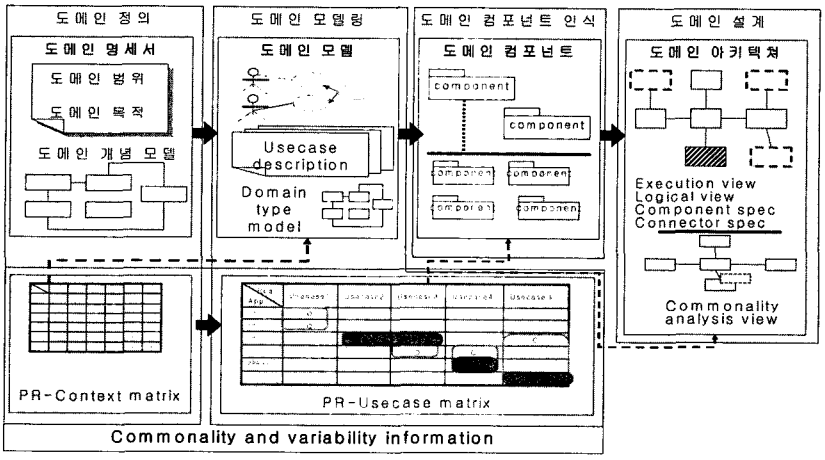

그림 11 도메인 분석, 설계 과정의 산출물 관계

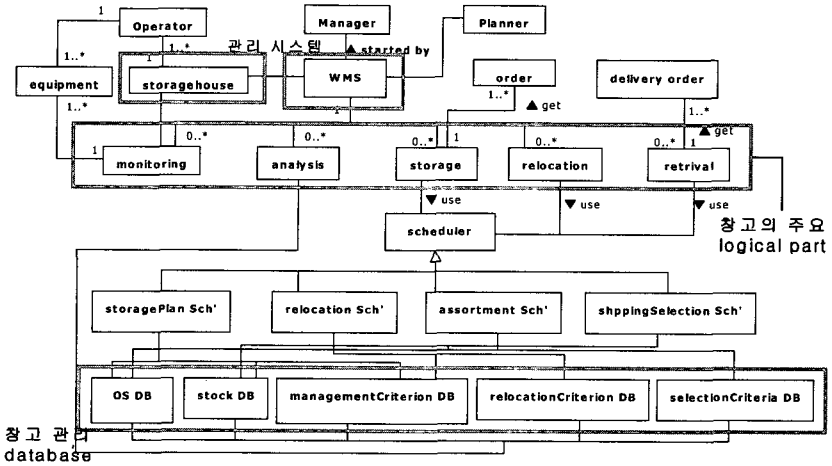

그림 12 후판 창고관리 시스템 도메인 개념 모델

그림 13은 3.1.3 단계 수행 과정에서 생기는 PR-Context 메트릭스의 일부 모습을 보여준다. 메트릭스의 가장 왼쪽 컬럼은 도메인 내에 존재하는 레거시 시스템의 요구사항들과 시나리오를 분석하여 나온 primitive requirement를 나열하게 된다. 가장 상위에는 도메인 내의 각기 다른 시스템들이 놓이게 된다. 시스템들은 각각 자신의 요구사항들을 이미 나누어 놓은 primitive requirement들을 확인하여 분석을 하게 된다. 이 과정을 반복적으로 적용함으로써 같은 환경에서 같은 요구사항들을 가진 일련의 시스템들은 같은 문맥(context)으로 묶을 수 있다. 후판 창고 관리 시스템 도메인에서는 'Login', 'Log off', 'Get rolling', 'received a notice of warehouse'와 같은 요구사항들은 공통성을 가진 요구사항으로 추출된다. 또한 여러 가지 스케줄러는 모든 시스템에서 항상 가지고 있는 요소이지만 그 특성상 내용의 변화 가능성을 가지기 때문에 다양성 속성으로 규정된다.

4.2 후판 창고관리 시스템 도메인 모델

그림 14는 유즈케이스의 공통성과 다양성을 추출하기 위한 과정에서 산출되는 PR-Usecase 메트릭스를 보여준다.

이 메트릭스는 유즈케이스가 어떠한 요구사항들로 이루어 졌는지를 보여주는 것으로 이를 참조하여 유즈케이스에 공통성과 다양성의 성질을 부여하게 된다. 유즈케이스는 관련 요구사항 중심으로 묶어주는 기능을 가지기 때문에 유즈케이스가 겹치는 경우는 피할 수 있다. 그러나 하나의 요구사항들이 여러 유즈케이스에 걸쳐 나오는 경우는(crosscutting) 이를 따로 분리하여 독립된 유즈케이스를 만들고 이를 <<include>>관계로 연

Context/system / Primitive requirements	A steel com. context1	B steel com. context 2	C steel com. context	...	후판 system Sn
Login	O	O	O		O
Login off	O	O	O		O
Get rolling	O	O	O		O
Get a basic storage	O	O	X		O
Operate a storage scheduler	Ov	Ov	X		Ov
Serve a storage notice	O	O	O		O
Get a shipping document for delivery	O	O	O		O
Operate a shipping scheduler	Ov	Ov	Ov		Ov
Issue a deliver order	O	O	O		O
Operate a relocation scheduler	Ov	Ov	Ov		Ov
Monitor warehouse state	O	X	O		O
Monitor equipments	X	O	X		X
Manage operation of trucks	O	X	X		O
⋮					

O : 요구되는 PR
X : 요구되지 않는 PR
Ov: 요구되지만 다양성의 속성을 가진 PR
▭ : *variable primitive requirement*
⬚ : optional primitive requirement

그림 13 PR-Context 메트릭스

Primitive requirements	Property	Storage Usecase (system)	PlanBasic Storage Usecase	planFirst Storage Usecase	planSecond Storage Usecase	storage Usecase
Get a order sheet	P	O	O			
Analyze a order sheet	Pv	O	O			
Plan a basic storage	P	O	O			
Get a rolling order	C	O		O		
Plan a first storage	C	O		O		
Operate storage scheduler	Cv	O		O	O	
Register storage plan on DB	C	O		O		
Display positions of first storage	C	O		O		
Get a storage order	C	O				O
Plan a second storage	P	O			O	
Display positions of second storage	P	O			O	
Serve a storage notice	C	O				O
⋮						O

optional Usecase
variable Usecase
Storage Scheduler Usecase
crosscutting

C: common PR
P: optional PR
Cv: common PR with variables
Pv: optional PR with variables

그림 14 PR-Usecase 매트릭스

결시킨다.

그림 14에서 'planBasicStorage'와 'planSecond-Storage' 유즈케이스가 가지고 있는 primitive requirement들은 모두 선택성을 가지고 있기 때문에 선택적 유즈케이스로 구분된다. 또한 'planFirstStorage'와 'planSecondStorage' 유즈케이스에 스케쥴링을 사용하는 요구사항이 겹쳐 나타나기 때문에 이를 따로 분리하여 독립된 유즈케이스로 만들고 이것은 다양성의 속성을 가지고 있기 때문에 다양성 유즈케이스로 구분된다. 겹쳐진 부분을 제외한 'planFirstStorage'와 'storage' 유즈케이스는 공통성 유즈케이스가 된다. 그림 15는 이러한 과정을 거쳐 구분된 유즈케이스의 속성들을 도식적으로 표현한 입고 유즈케이스 다이어그램을 보여준다.

유즈케이스 다이어그램을 만드는 과정과 메트릭스를 만드는 과정은 서로 반복적으로 병행하여 일어나면서 속성을 가진 도메인 모델이 완성된다.

4.3 후판 창고관리 시스템 도메인 컴포넌트

도메인 컴포넌트는 이전 단계에서 생성된 도메인 모델을 기반으로 이루어진다. 그림 16은 입고 유즈케이스를 바탕으로, 3.3.1 도메인 컴포넌트 추출의 수행 결과이다.

그림 17은 3.3.2 단계의 결과물이다.

추출된 하나의 도메인 컴포넌트의 속성을 확인하게 되고 이때 속성이 혼합되어있는 경우 이를 분리 가능한지를 판단하여 일반화 형태로 재구성한다. 여기서는 도메인 컴포넌트 내부에 존재하고 있는 'Schedule for Storage' 컴포넌트가 다양성을 포함하면서 독립 가능하

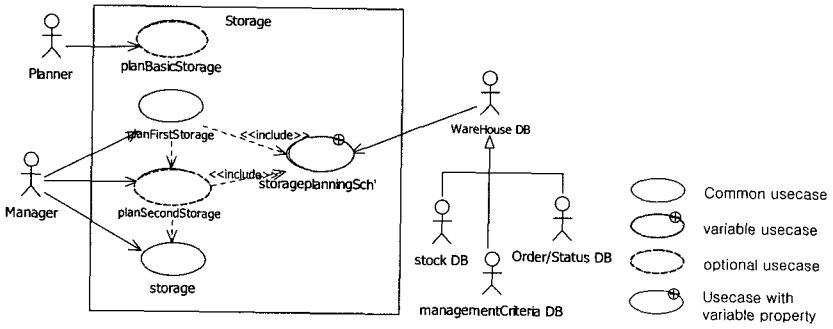

그림 15 입고 도메인 유즈케이스 모델

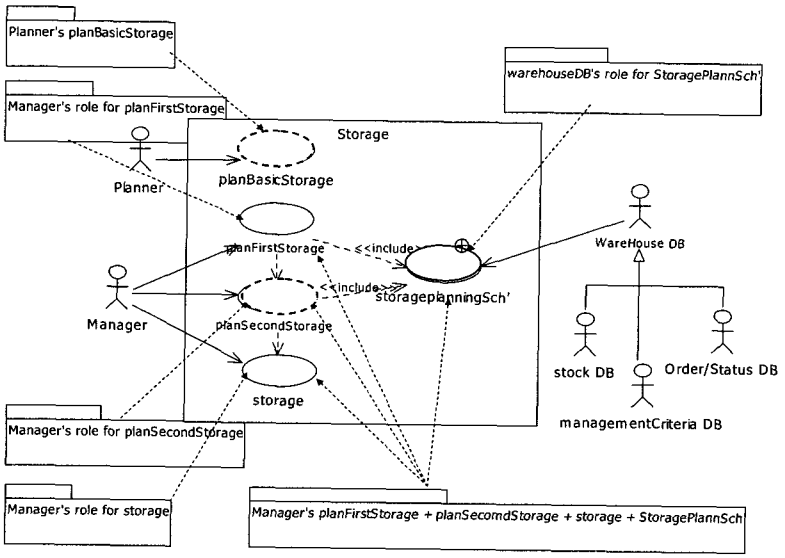

그림 16 추출된 입고 관련 도메인 컴포넌트

므로 하나의 컴포넌트로 분리시키고 그 속성을 다양성 컴포넌트로 부여한다. 또한, 'secondStoragePlanning' 클래스가 선택적 속성을 가지고 독립 가능하므로 하나의 선택성 속성을 가진 컴포넌트로 분리되었다.

4.4 후판 창고관리 시스템 도메인 아키텍쳐

추출된 도메인 컴포넌트들 간의 관계를 인식하여 도메인 아키텍쳐로 표현한다. 그림 18은 입고 요구사항을 수행하기 위해 발생하는 컴포넌트간의 상호작용을 표현하는 실행 뷰를 교류도(sequence diagram)를 통해 보여준다. 이를 바탕으로 컴포넌트의 오퍼레이션을 찾을 수 있다.

그림 19는 컴포넌트 명세서에 나타나는 포트의 내용 중 각 오퍼레이션에 관한 정보를 나타내기 위해, 입고에 관련한 requestRolling() 오퍼레이션을 보여준다.

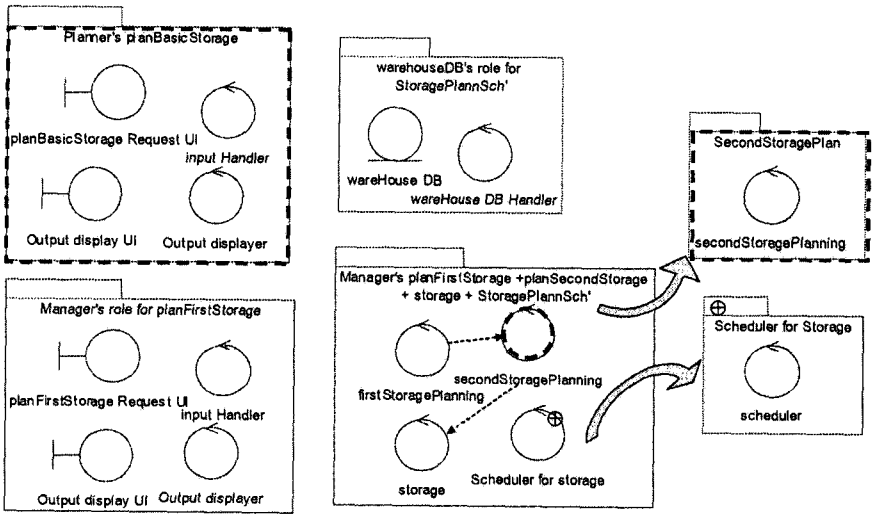

그림 17 도메인 컴포넌트의 재구성 과정

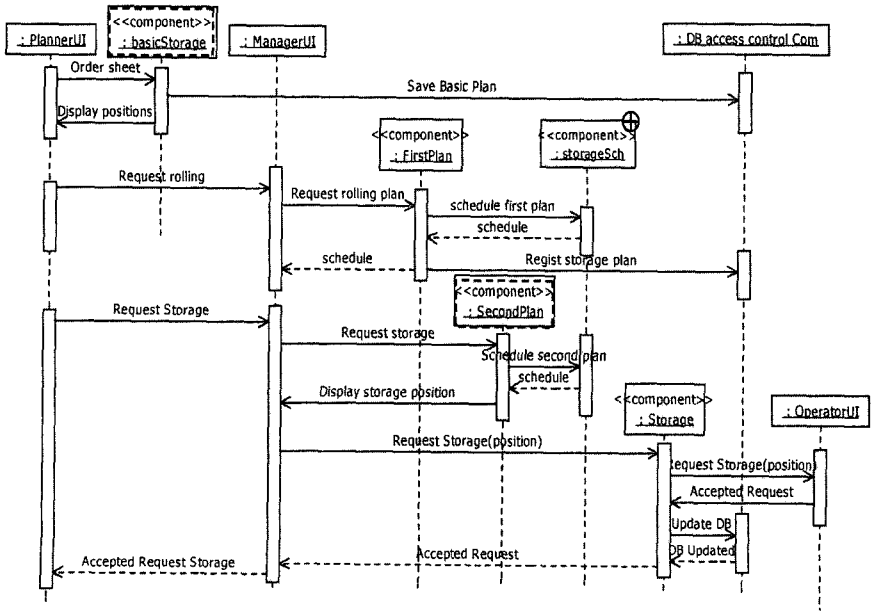

그림 18 입고 관련 교류도

Name	request Rolling (rolling: rolling)
Responsibilities	압연지시를 받아서 압연지시 DB에 저장한다.
Type	System
Cross References	Use cases: planFirstStoragestem
Notes	Rolling은 압연번호, 치수, 수량, 납기를 포함하는 type이다.
Exceptions	DB에 문제가 발생할 경우 오류 메시지를 리포트한다.
Pre-conditions	자재관리 시스템이 압연지시를 보낸 상태이다.
Post-conditions	압연지시 DB가 update 되었다.

그림 11 입고 관련 오퍼레이션 기술

5. 결론 및 향후 연구 과제

수십 년 전부터 소프트웨어 공학 분야에서 많은 관심을 가지고 연구되어온 분야가 소프트웨어 재사용이다. 실생활에서와 마찬가지로 '재사용'을 한다는 것은 경제적인 측면이나 시간적인 측면에서 많은 이점을 가져다준다. 재사용이 효율적으로 수행되기 위하여 체계적인 방법이 필요하며 이를 위해 관련된 시스템의 집합이라는 도메인에서 공통성과 다양성을 찾는 분석을 하게 된다. 지금은 더군다나 컴포넌트라는 새로운 패러다임의 대두로 재사용에 대한 관심은 더욱더 높아졌으며 이와 더불어 컴포넌트 기반 개발 방법에 적합한 도메인 분석 방법이 필요하게 되었다.

본 연구에서는 기존의 방법과는 달리, 컴포넌트 기반 소프트웨어 개발 프로세스에 적합하게 상호 작용할 수 있는 도메인 분석, 설계 방법을 제시하였다. 즉, 기존의 연구에서는 도메인 컴포넌트를 인식하고 도메인 아키텍처를 이용해서 그들간의 관계를 고려하여 컴포넌트를 조립하는 과정에 대한 정보를 얻을 수가 없었다. 그러므로 본 연구에서는 컴포넌트 기반 소프트웨어 개발 시 필요로 하는 컴포넌트를 인식, 선택하고, 컴포넌트간의 연결 관계, 커넥터 정보를 도메인 엔지니어링 과정을 통해서 산출하게 함으로써 컴포넌트 기반 소프트웨어 개발 프로세스를 지원하게 하였다. 또한 도메인 분석에서 가장 중요한 공통성과 다양성을 인식하는 데 있어서 지금까지의 연구들은 도메인 전문가의 경험과 직관성에 의존하도록 하고 있었다. 그러나 본 연구에서는 공통성과 다양성을 찾는 방법을 요구사항을 일반화시키는 과정을 통해 그 정보를 객관성 있게 추출할 수 있도록 하였으며, 이 정보를 메트릭스 형태로 만들어 계속 유지, 정제시켜가며 각 단계에 이용함으로써 공통된 유즈케이스, 공통된 컴포넌트를 발견할 수 있도록 하였다. 또한 각 단계의 산출물간의 관계를 좀 더 명확히 나타냄으로써 프로세스 진행에서 발생하는 모호성을 줄이고자 노력하였다.

향후 연구 과제로는 두 가지 방향으로 나아가게 된다. 첫째는 도메인 아키텍처를 이용한 도메인 컴포넌트의 구현 가능성을 검토해보고 이에 적용될 수 있는 개념과 기술들에 대해 연구한다. 둘째는 제시한 도메인 분석과 설계 방법을 이용하여 컴포넌트 기반 소프트웨어 개발에 적용되는 과정에 대하여 연구가 필요하다. 이로써 처음에 제시한 전체 연구 프로세스가 완성되게 될 것이다.

참 고 문 헌

[1] Creps D., Klingler, C., Levine, L., and Allemang, D., "Organization Domain Modeling(ODM) Guidebook Version 2.0", Software Technology for Adaptable, Reliable Systems (STARS), 1996.

[2] Frank, S., "The Three "R's" of Mature System Development: Reuse, Reengineering, and Architecture", In The Fifth Systems Reengineering Technology Workshop 1995. URL: http://source.asset.com/stars/lm-tds/Papers/sysdev/nswc-95.html

[3] Klingler, C. D., "DAGAR: A Process for Domain Architecture Definition and Asset Implementation.", In: Proceedings of ACM TriAda, 1996.

[4] Kang, K. C., "Feature-Oriented Domain Analysis for Software Reuse", Joint Conference on Software Engineering, pp. 389-395, 1993.

[5] SEI in Carnegie Mellon University, "Feature-Oriented Domain Analysis", URL:http://www.sei.cmu.edu/str/descriptions/foda_body.html

[6] Kang, K. C., Kim, S., Lee J.,and Kim, K., "FORM: A Feature-Oriented Reuse Method with Domain Specific Reference Architectures", Pohang University of Science and Technology(POSTECH), 1998.

[7] Griss, M. L., Favaro, J., and d'Alessandro, M., "Integrating Feature Modeling with the RSEB", in Proceedings of 5th International Conference on Software Reuse, Victoria Canada, June, IEEE, pp. 76-85, 1998.

[8] van Gurp, J., Bosch, J., and Svahnberg, M., "On the notion of variability in software product lines", Proceedings on Working IEEE/IFIP Conference on Software Architecture, pp. 45-54, 2001.

[9] Frakes, W., Prieto-Diaz, R., and Fox, C., "DARE-COTS: A Domain Analysis Support Tool", Proceedings on XVII International Conference of the Chilean Computer Science Society (Valparaiso, Nov, 1997), pp. 73-77, 1997.

[10] SEI in Carnegie Mellon University, "Component-Based Software Development/COTS Integration", http://www.sei.cmu.edu/str/descriptions/cbsd_body.html

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

[12] Harmon, P., "Visual Modeling Tools, Case Vendors, and Component Methods", Component Development Strategies, Volume IX, No. 5, 1999.

[13] Short, K., "Component Based Development and Object Modeling", Sterling Software, 1997.

[14] Jacobson, I., Booch, G., and Rumbaugh, J., "The Unified Software Development Process", Addison-Wesley, January 1999.

[15] John, C. and John, D., "UML Components", Addison-Wesley, October 2000.

[16] 한국전자통신연구원, "마르미-III 방법론", URL:http://www.component.or.kr

[17] Digre T., "Business Object Component Architecture", IEEE software Vol.15, No.5, September/October, pp. 60-69, 1998.1.

문 미 경

1990년 2월 이화여자대학교 전자계산학과(학사). 1992년 2월 이화여자대학교 전자계산학과(석사). 2000년 3월~현재 부산대학교 컴퓨터공학과 박사과정. 1998년 3월~1999년 2월 안산1대학 사무자동학과 겸임교수. 관심분야는 도메인 공학, 컴포넌트 기반 소프트웨어 개발, 객체 지향 방법론, 도메인 분석 및 아키텍처 개발 등

박 준 석

1999년 8월 부경대학교 컴퓨터공학과(학사). 2002년 2월 부산대학교 컴퓨터공학과(석사). 2002년 3월~현재 부산대학교 컴퓨터공학과 박사과정. 2002년 11월~현재 (주)사이버맵월드 부설연구소 연구원. 관심분야는 소프트웨어 아키텍처, 컴포넌트 기반 소프트웨어 개발, MDA(Model Driven Architecture), 도메인 공학 등

염 근 혁

1985년 서울대학교 계산통계학과(학사) 1992년 Univ. of Florida 컴퓨터공학과(석사). 1995년 Univ. of Florida 컴퓨터공학과(박사). 1985년 1월~1988년 2월 금성반도체 컴퓨터연구실 연구원. 1988년 3월~1990년 6월 금성사 정보기기연구소 주임연구원. 1995년 9월~1996년 8월 삼성SDS 정보기술연구소 책임연구원. 1996년 8월~현재 부산대학교 컴퓨터공학과 부교수. 부산대학교 컴퓨터 및 정보통신 연구소 연구원. 관심분야는 컴포넌트 기반 소프트웨어 개발, 도메인 공학, 소프트웨어 아키텍처, 객체지향 소프트웨어 개발방법론, 요구 검증, 분산 컴포넌트, 소프트웨어 재사용 등