패턴 기반 소프트웨어 개발을 위한 효과적인 패턴 선정 프로세스

(An Effective Pattern Selection Process for Developing of Pattern Based Software)

최 진 명 † 류 성 열 ††
(Jin Myung Choi) (Sung Yul Rhew)

요 약 디자인 패턴은 지난 10년 이상 소프트웨어 공학 영역에서 활발하게 연구되어 여러 유형의 디자인 패턴이 정의되었다. 그러나 이들 패턴은 자료구조와 알고리즘에 비해 소프트웨어 개발 과정에 빈번하게 사용되지 못하고 있다. 더욱이 CBD96, RUP, MaRMI III 같은 CBD 방법론들은 분석, 설계, 개발 과정 중에 디자인 패턴을 선정하고 적용하는 방법이 언급되어 있지 않다. 본 논문에서는 GoF, J2EE 패턴 카타로그에 제시된 디자인 패턴을 중심으로 응용 소프트웨어를 개발하기 위해 분석, 설계, 개발 단계에 적용되는 디자인 패턴을 효과적으로 선정하는 프로세스를 제시한다. 제시된 프로세스를 항공업무 어플리케이션 개발 과정에 적용하여 패턴 기반 소프트웨어를 개발해 나가는 효과적인 방법을 보이고 RUP와의 차이점 및 유용함을 제시한다.

키워드 : 디자인 패턴, 프로세스, 방법론, 컴포넌트

Abstract Over the past decade, several types of design pattern have been defined in the software engineering area. But these patterns have not been used so often compared with data structure and algorithm. Likewise, methods to select and apply design patterns during analysis, design, and development are not mentioned in CBD methodologies such as CBD96, RUP, and MaRMI III. This paper suggests the process of effectively selecting design patterns which can be applied to analysis, design, and development for development of application software with based on those offered by GoF and J2EE pattern catalogs. It also demonstrates how to effectively pattern-based software and shows differences and relative advantages from RUP by applying the suggested process to the development of aviation job application.

Key words : Design Pattern, Process, Methodology, Component

1. 서 론

오늘날 소프트웨어는 규모가 커지고 운영환경이 분산환경 및 웹과 같은 월드와이드 환경으로 확장되면서 개발이 복잡해지고 여러 개발 인력과 장시간의 개발시간을 필요로 하게 되었다. 이런 소프트웨어 개발추세의 변화를 수용하기 위해 새롭고 다양한 소프트웨어 개발 기법 - 방법론, 아키텍처, 개발 언어, 도구(Tools) 등 - 이 연구되고 개발되었다.

코드의 재사용을 강조하는 객체지향 언어, 웹기반의 소프트웨어를 개발할 수 있는 서버측 언어 및 스크립트 언어들이 개발언어의 주류를 이루게 되었고, 개발 방법의 측면에서도 단순한 소스코드의 재사용이 아닌 실행 모듈 레벨에서의 재사용을 지원하고 특정 기능 및 비즈니스 영역을 재사용할 수 있는 컴포넌트 중심의 소프트웨어 개발을 가능하게 하는 CBD96, RUP, Catalysis, FUSION, MaRMI III 같은 소프트웨어 개발방법론이 개발되었다. 이들 방법론들은 소프트웨어의 변경 가능성, 재사용성, 유지보수성 등을 고려하여 개발해 나가도록 가이드하고 있다. 그러나 이런 진보된 개발 언어와 개발방법론을 통해 소프트웨어를 개발하더라도 소프트웨어 개발 생산성의 향상, 만족할 만한 수행 성능, 안정성, 신뢰성 등을 보장하지 못하는 것이 현실이다. 또한 방법론을 적용해서 소프트웨어를 개발한다 할지라도 개발과정에서 빈번하게 발생하는 문제들이 있다. 여기에는

· 본 연구는 숭실대학교 교내 연구비 지원으로 이루어졌음
† 정 회 원 : 숭실대학교 컴퓨터학과
jinmyung@shinbiro.com
†† 종신회원 : 숭실대학교 컴퓨터학부 교수
syrhew@comp.ssu.ac.kr
논문접수 : 2004년 3월 5일
심사완료 : 2005년 3월 15일

소프트웨어 개발자들이 개인의 경험에 의지해서 소프트웨어를 개발하려 하고, 기술적으로 숙련되지 못한 개발자가 개발에 참여하며, 고객에 의해 요구사항이 빈번하게 변경되고, 개발 과정에서 개발 인력이 자주 교체된다는 점, 그리고 정형화된 알고리즘이나 특정 문제에 대한 공통된 해법이 반영되지 않는 문제 등이 있다. 이런 문제점들을 보완하고 해결하기 위한 방법이 디자인 패턴의 적용이다. 디자인 패턴은 빈번하게 발생되는 문제들에 대해 오랜 기간 동안의 여러 경험에 의해 공통되고 일관성 있는 정확한 해결책을 제시한다[1-3].

프레임워크는 이들 패턴들이 적절하게 적용된 문제 도메인 및 문제 도메인에 대한 해법을 내포하고 있어 개발자들이 개발단계에서 쉽게 사용할 수 있도록 하고 있다[4]. 그러나 IBM의 샌프란시스코 프레임워크와 같은 프레임워크 내에는 불필요한 문제 도메인과 그에 대한 해법까지도 포함하고 있어 개발자가 적절한 기능 함수를 사용하기에 어려움이 있고, 개발된 소프트웨어의 크기가 증가되며, 프레임워크에 대한 기능적 의존성이 커진다. 프레임워크가 패턴을 내포하고 있다 할지라도 요구사항에 대한 분석 및 설계 단계에서 패턴을 고려하지 않은 상태로 소프트웨어 개발을 진행하면 프레임워크가 제공하는 기능들을 사용할 때 패턴의 중복 및 불필요한 패턴까지도 사용하게 되어 소프트웨어의 성능을 저하시킨다. 그리고 소프트웨어를 개발하는 개발자들은 프레임워크에서 제공하는 기능 함수만을 사용하므로 디자인 패턴에 대한 지식, 사용상의 장점 및 사용 경험 등을 습득할 수 없다. 이와 같은 이유로 많은 개발자들이 디자인 패턴을 공부하고 있고, 그것을 소프트웨어의 분석, 설계 및 개발에 적용하려고 노력하고 있다. 그럼에도 불구하고 현재의 많은 소프트웨어 개발 프로젝트에서 디자인 패턴이 실질적으로 적절하게 적용되고 있지 못하다는 것이 현실이다. 그 이유는 효과적이고도 유용한 많은 방법론들이 그들 내에 디자인 패턴을 정의하고 있지 않고, 어떻게 디자인 패턴을 식별하고, 어떻게 분석, 설계, 개발에 적용해야 하는 가에 대한 가이드가 없기 때문이다. 실제 소프트웨어 설계와 개발 단계에 디자인 패턴을 적용하기 위해서는 패턴 각각의 문제영역과 그 문제영역을 해결하기 위한 방법, 패턴간의 연관 관계, 패턴을 적용할 카테고리 분류 작업 등이 필요하다. 이에 본 논문에서는 이러한 필요성을 충족시키며 원활하게 패턴을 적용한 소프트웨어를 개발할 수 있는 패턴 선정 기법을 제시한다. 먼저 2장에서는 관련 연구로서 디자인 패턴을 정의하고, 'GoF 디자인 패턴 카타로그'[1], 'J2EE 디자인 패턴 카타로그'[2]를 제시하여 패턴의 유형 및 관계를파악한다. 그리고 RUP 방법론의 일반적 개발 프로세스를 제시하여 패턴 선정 기법이 적용되어야 할 프로세스 상의 적절한 단계를 식별한다. 3장에서는 요구사항 분석, 아키텍처 선정, 시스템 계층화, 패턴 매핑 등의 패턴 선정 기법의 세부 절차 및 방법을 제안한다. 4장에서는 항공회사의 특정 항공업무 소프트웨어 개발에 제시한 기법을 적용한 사례와 효과, 기존에 존재하던 타 프로세스와의 차이점 및 특이성을 제시하고 5장에서 결론을 맺는다.

2. 관련 연구

2.1 디자인 패턴

디자인 패턴은 특정 영역의 문제가 주어졌을 때 하나 또는 여러 개의 패턴들이 어떻게 책임이 객체에 할당되어야 하고 객체와 객체간에는 어떤 관계가 존재해야 하는가에 대한 지침을 제공한다. 그러나 소프트웨어를 개발하는 과정에 디자인 패턴이 적절하게 사용되고 있지 못하다. 그 이유는 패턴이 무엇이고 패턴은 어떤 특성을 갖고 있는지를 분석/설계자, 개발자가 정확하게 이해하고 있지 못하기 때문이다.

디자인 패턴에 대한 정의는 개개의 클래스, 인스턴스, 컴포넌트들의 상위 단계인 추상 개념을 확인하고 특정 짓는 것[1], 반복되는 구조에 대한 디자인 주제의 재사용성에 좀 더 초점을 두는 것[5]이다. 그리고 패턴에는 9가지 특성이 있다[6]. 패턴은 여러 개발자들에 의해 오랜 기간 동안 개발 과정에서 겪었던 문제와 그 문제를 효과적이면서 정확하게 해결하기 위해 고민하고 찾아낸 경험 및 관용적 해법의 산물이다. 따라서 이런 디자인 패턴 각각은 문제영역과 해법영역 형태의 쌍을 이루고 있어, 각 패턴이 다양한 문제 영역에서 어떻게 그 문제를 해결해 나가는 가에 대하여 조언하고 효과와 여러 영향을 고려한다.

디자인 패턴을 적용하는 것의 장점은 첫째, 요구사항의 분석, 설계를 명확하게 할 수 있고, 소스 코드에 대한 리팩토링(Refactoring)이 가능하게 하여 프로그램 소스 코드 상의 모호성과 중복을 제거하여 소프트웨어를 쉽고 유연하게 확장할 수 있다. 둘째, 객체지향 분석, 설계 시 공통된 모델링 언어로 UML이 사용되는 것과 같이, 클래스 설계 시 공통된 설계 언어로써 사용될 수 있다. 즉, Abstract, Factory, Adaptor 등 일련의 패턴을 사용함으로써 클래스의 형태, 클래스간의 관계 - 상속, 연관, 일반화 - 등을 명확하게 표현하여 누구나 쉽게 이해하고 코드를 재사용 할 수 있다. 셋째, [7]에서와 같이 패턴을 이용한 리팩토링 방법을 통해 객체지향 언어로 이미 작성되어 있는 레거시(Legacy) 시스템을 효과적으로 재공학(Reengineering) 할 수 있다. 이와 같은 장점으로 새로운 패턴이 다양한 문제 영역에 대한 해법을 제공하기 위해 꾸준히 연구되고 있고, EJB, COM+와

같이 프로그램의 재사용을 용이하게 하는 컴포넌트 기반 소프트웨어 개발(CBD)과 진화 가능한 소프트웨어(Evolutionary Software)의 개발에 실질적이고도 유용하게 사용되고 있다.

2.2 RUP의 소프트웨어 개발 프로세스

RUP는 객체지향 소프트웨어의 개발과 컴포넌트 기반 소프트웨어를 개발하기에 적합한 소프트웨어공학적 프로세스를 내포하고 있다. RUP는 또한 객체지향 분석, 설계 시 사용되는 모델링 언어인 UML을 이용하고 있어 방법론의 이해와 사용을 위한 접근이 용이하다. '도입(Inception)→상세화(Elboration)→구축(Construction)→이행(Transition)'의 생명 주기를 갖는 RUP는 시간에 따른 변화를 효과적으로 만족시키기 위해 동적인 생명주기, 핵심적인 작업흐름(workflow)과 같은 2차원적 구조를 갖는다. 생명 주기의 각 단계에서 본질적으로 수행해야 할 활동과 이정표는 표 1과 같다.

표 1 RUP 생명 주기별 활동 사항 및 이정표

단계	활동 사항	이정표
도입	- 프로젝트 범위 산정 - 비즈니스 사례의 계획과 준비 - 후보 아키텍처 종합 - 프로젝트를 위한 환경 준비	생명주기 목표
구체	- 아키텍처 정의, 검증, 기준 설정 - 추구하는 비전 정제 - 구축 단계에서의 자세한 반복 계획 생성 및 기준 설정 - 개발 사례 정제 및 개발 환경 구축 - 아키텍처 정제 및 컴포넌트 선택	생명주기 아키텍처
구축	- 자원 관리, 통제, 프로세스 최적화 - 컴포넌트 개발 완료 및 정의된 평가 기준에 의한 테스트 - 비전에 맞춘 수용 기준 대비 발표할 제품 평가	초기 운영 능력
인도	- 배치 계획 실행 - 최종 사용자 지원 자료 완성 - 개발 사이트에서 배포할 제품 테스트 - 제품 완제품 생성 - 사용자 피드백 수렴 - 피드백에 기준한 제품 튜닝 - 사용자에게 가치있는 제품 작성	제품 발표

핵심적인 작업흐름(workflow)은 가시적으로 가치 있는 결과를 생산하는 순차적인 작업활동(activity)을 의미한다. 작업흐름은 UML의 Sequence 다이어그램, Collaboration 다이어그램, Activity 다이어그램으로 표현할 수 있는데 RUP는 Activity 다이어그램을 사용하여 표현한다. 그리고 RUP는 작업자와 활동을 논리적으로 분류시켜 6개의 핵심 엔지니어링 작업흐름과 3개의 핵심 지원 작업흐름 등 총 9개의 핵심 작업흐름(core workflow)으로 구분된다[8].

RUP는 '분석→설계→개발→테스트' 등의 개발 전 과정을 점진적으로 기능 확장을 하면서 반복적으로 개발하는 프로세스를 갖는다[8-10]. 따라서 RUP를 이용해서 소프트웨어를 개발하면 개발 과정에서 발생될 일련의 행위들에 대해 예측가능하고, 변경에 민감하며, 사용자(고객)를 적극적으로 참여시킬 수 있고, 개발 중간 단계에서부터 조기에 위험을 감지하고 해결해 나가므로 위험을 최소화 하여 결과적으로 개발 생산성이 높고 품질이 높은 소프트웨어를 개발할 수 있다.

2.3 POAD

컴포넌트와 같은 블록을 생성하여 소프트웨어를 개발하고자 할 때 소프트웨어의 디자인 단계에서 컴포넌트를 효과적으로 설계하기 위한 접근법이 POAD(Pattern-Oriented Analysis and Design)이다[11,12]. 이 프로세스는 패턴의 컴포지션과 인터페이스를 이용해서 소프트웨어를 설계하는 방법을 제시한다. POAD는 그림 1과 같이 3개의 단계와 8개의 활동으로 구성되어 있다. 요구사항 분석 활동에서는 어플리케이션 요구사항을 분석하여 필요한 컴포넌트들을 식별한다. 이후 패턴 선정 활동에서는 각각의 컴포넌트에 대해 식별된 컴포넌트의 기능성 및 책임을 고려해서 소규모 패턴 라이브러리로부터 설계 패턴들을 선정한다. 이렇게 해서 분석단계가 끝난 후 패턴 중심의 설계 모델을 작성한다. 먼저 패턴 레벨 모델 작성 활동에서는 분석단계에서 선정된 패턴들을 중심으로 여러 패턴들이 개별 컴포넌트에 어떻게 사용되는 가에 대한 모델을 작성한다. 이 패턴 모델은 패턴 인스턴스와 각 패턴간의 관계가 정의된 패턴 레벨 다이어그램으로 표현된다. 다음으로는 각 패턴의 인터페이스를 식별하고 식별된 인터페이스를 통해 패턴들이 연계되는 패턴 다이어그램을 작성한다.

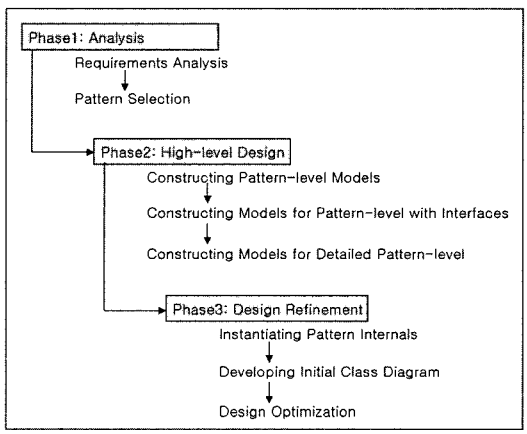


그림 1 POAD 프로세스

이 때는 패턴 인터페이스간의 관계를 식별해서 다이어그램에 표현한다. 설계단계의 마지막 활동인 상세 패턴 레벨 모델 작성 활동에서는 컴포넌트에 적용될 패턴 및 패턴의 인터페이스를 초기 클래스 다이어그램 형태로 작성한다. 설계 정제 단계의 패턴 내부 실제화 활동에서는 패턴에 의해 모델링된 컴포넌트의 내부 설계 모델에 대해 실제 개발 단계에서 사용될 수 있도록 패턴의 인스턴스 및 인터페이스에 대해 의미 있는 이름을 부여하고 개념적으로 이들 패턴들을 특성화하고 분류하고 교정한다. 이 때 패턴을 분류하기 위해 [13]에서 Keller에 의해 제시된 Concretization, Specialization, Scoping, Revision의 분류를 적용한다. 초기 클래스 다이어그램 개발 활동에서는 상세 패턴 레벨 다이어그램으로부터 패턴 내부에 대해 실제화한 모델과 패턴 인터페이스를 사용하여 UML 클래스 다이어그램을 작성한다. 이렇게 개발된 클래스 다이어그램은 컴포넌트의 정적 설계 모델을 개발하기 위한 시작단계가 된다. 작성된 UML 클래스 다이어그램에는 여러 패턴 및 패턴의 인터페이스를 모델링 했기 때문에 추상클래스나 서브 클래스들의 반복된 사용이 발생되므로 최종활동인 설계 최적화 활동에서 축소, 병합, 그룹핑 등을 통해 간결하면서도 구체적인 클래스 다이어그램으로 최적화한다.

3. 패턴 선정 프로세스

패턴 기반 소프트웨어를 개발함에 있어 가장 중요한 것은 분석 및 설계 단계에서 도출된 문제영역에 대해 어떻게 가장 적합한 패턴을 선정해서 적용하는 가이다. 패턴 기반 소프트웨어를 개발하기 위해 본 논문에서 제시하는 패턴 선정 프로세스의 주된 목표는 요구사항 분석을 통한 적용할 기반기술 선정, 소프트웨어/하드웨어 아키텍처 선정, 분석된 목표 시스템의 계층구조에 맞도록 디자인 패턴을 그룹화 하는 것이다. 이를 수행하기 위한 프로세스는 그림 2와 같은 4개의 단계(Phase)로 구성된다.

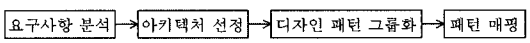


그림 2 시스템 관점의 패턴 선정프로세스

요구사항 분석 단계에서는 사용자의 요구사항 분석 결과를 기반으로 현행 시스템 또는 목표시스템을 분석해서 패턴 기반 소프트웨어 개발에 필요한 핵심 기술을 선정한다. 아키텍처 선정 단계에서는 요구사항 분석 단계에서 분석된 시스템을 설계하기 위해 선정된 기술을 검증한 후 하드웨어/소프트웨어 아키텍처를 선정한다. 그리고 이 단계에서 목표 시스템을 위해 선정한 하드웨어/소프트웨어 아키텍처를 검증한다. 시스템 계층화 단계에서는 선정된 아키텍처를 기반으로 모델링한 시스템을 [2]에서 제시한 6개 계층으로 나눈다. 다음으로 [1], [2]에 정의되어 있는 디자인 패턴 및 새롭게 발표되어 사용되고 있는 디자인 패턴들을 수집한다. 패턴 매핑 단계에서는 설계된 시스템 아키텍처와 관련된 패턴을 분류하여 이 패턴들을 나누어진 6개의 계층에 적절하게 매핑시킨다. 이때 매핑에서 주의할 점은 반드시 각 계층의 특성과 매핑하고자 하는 패턴의 문제영역(Problem), 해법(Solution), 그리고 적용 결과(Consequence)를 정확하게 숙지하고 있어야 한다. 프로세스 각 단계별 세부 활동(Activity) 및 프로세스 진행 흐름도는 그림 3과 같다.

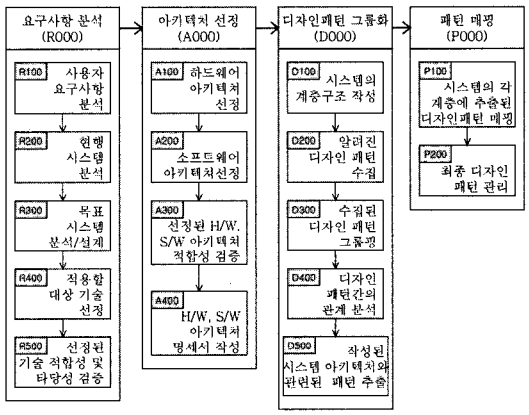


그림 3 시스템 관점의 상세 패턴 선정 프로세스

3.1 요구사항 분석(R000)

- 사용자 요구사항 분석(R100) - 개발할 패턴기반 소프트웨어의 기능적, 비기능적 요구사항과 예외사항 등을 분석한다.
- 현행 시스템 분석(R200) - 이미 운영되고 있는 현행 시스템이 있는 경우에 대해 수행하고, 현행 시스템이 없다면 이 활동은 생략한다. 이 활동에서는 현행 시스템의 물리적 아키텍처와 소프트웨어 아키텍처, 물리적 네트워크 환경 정보 등을 분석한다.
- 목표시스템 분석/설계(R300) - 현행 시스템이 있다면 현행 시스템 분석 활동의 결과물을 토대로 분석된 문제점, 개선사항, 신규 요구사항 등을 고려하고, 현행 시스템이 없다면 요구사항 분석 활동에서의 비기능적 요구사항, 예외사항 등을 고려하여 개발할 목표시스템의 하드웨어 아키텍처, 소프트웨어 아키텍처 등을 분석, 설계한다.
- 적용할 대상 기술 선정(R400) - 프로그래밍 언어, 객체 또는 컴포넌트 기술, 유선 또는 무선 데이타 처리, 데이타 스트리밍, 보안 기술 등 개발할 목표시스템에 적용할 필수 및 선택적 기술사항을 선정한다.

• 선정된 기술 적합성 및 타당성 검증(R500) - 선정된 요소기술을 중심으로 구현 사례 수집, 핵심 기능군의 일부를 선행 개발하여 개발 시의 기술적 문제점, 개발에 드는 시간 및 인력 산정, 목표 시스템의 기능적 요구사항, 비기능적 요구사항의 만족도 측정 등을 통해 기술의 적합성 및 타당성을 검증한다.

3.2 아키텍처 선정(A000)

• 하드웨어 아키텍처 선정(A100) - 분석된 요구사항에서 보안관리, 장애 관리, 분산, 시스템 가용성, 시스템 안정성(장애 대책, 백업 대책), 확장성, 주전산기, 시스템 구성, 데이타베이스 및 디스크 구성 방안, 하드웨어/소프트웨어 사양 및 성능, 미들웨어 요구사항, 레이드(RAID)디스크, 네트워크 관리, 백업 장비 및 백업 체계, 기존 장비 등의 요구사항 정보를 토대로 하드웨어 아키텍처를 선정한다.

• 소프트웨어 아키텍처 선정(A200) - [14]에서 언급된 소프트웨어 아키텍처 구조 즉, 모듈(분할, 사용과 계층, 클래스), 컴포넌트와 연결(클라이언트/서버, 병행성, 프로세스, 공유 데이타), 할당(작업 지정, 배치, 구현)을 고려하여 소프트웨어 아키텍처를 선정한다.

• 선정된 H/W, S/W 아키텍처 적합성 검증(A300) - 선정한 하드웨어/소프트웨어 아키텍처에 요구사항 분석단계에서 선정된 기술을 적용하여 파일럿 프로젝트를 수행하거나 프로토타입을 실행하여 시스템의 수행성능, 응답속도, 하드웨어/소프트웨어 자원 점유율, 안정성 등을 검증한다.

• H/W, S/W 아키텍처 명세서 작성(A400) - 개요, 아키텍처 목표 및 제약 사항, [15]의 'UML 4+1 View'에 해당하는 Use Case View, Logical View, Process View, Batch View, Implementation View, 성능 및 품질 등의 항목으로 선정된 아키텍처에 대한 명세서를 작성한다.

3.3 디자인 패턴 그룹화(D000)

• 시스템의 계층구조 작성(D100) - 목표시스템을 표 2와 같이 5개의 계층으로 나눈다[2]. 클라이어트 계층은 실제 비즈니스 서비스를 요청하는 계층이고, 프리젠테이션 계층은 비즈니스 요청 및 전달받은 요청 결과를 보여주는 계층이다. 비즈니스 계층은 요청 받은 서비스에 대해 통합 및 자원 계층과 연계하여 비즈니스 기능을 수행한 후 그 결과를 프리젠테이션 계층을 통해 클라이언트 계층에 전달한다. 통합 계층은 비즈니스 기능을 수행하기 위한 외부 연계 및 인터페이스를 제공하고, 자원 계층은 서비스할 비즈니스 기능의 원천 데이타 및 기반 시스템으로 구성한다.

• 알려진 디자인 패턴 수집(D200) - 분석된 기능적 요구사항과 설계된 시스템 아키텍처를 기준으로 목표

표 2 시스템 5계층

계층	구성요소
클라이언트 계층	WEB 기반 클라이언트 어플리케이션, 애플릿, 독립 어플리케이션, GUI
프리젠테이션 계층	JSP, Servlet, 다른 UI 요소
비즈니스 계층	EJB, COM+, CORBA Component, DTO, 클래스
통합 계층	JMS, JDBC, ODBC, Connector, 레거시
자원 계층	데이타베이스, 외부 시스템, 레거시 자원

시스템을 개발하기 위해 디자인 패턴을 수집한다. 이때 수집하는 방법으로는 첫째, 알려져 있는 모든 패턴을 수집하는 방법과 둘째, 개인적 경험 또는 그룹원들의 경험을 토대로 한 선별된 패턴만을 수집하는 방법이 있다. 첫째 방법은 패턴을 수집하는 데 어려움이 있고 해당 패턴을 정확하게 사용하지 못할 수 있지만 요구사항, 아키텍처를 만족하는 패턴들을 최대한 많이 적용하여 보다 신뢰성 있는 검증된 패턴 기반 소프트웨어를 개발할 수 있다. 둘째 방법은 선별된 패턴들이 어느 문제영역에 사용되어야 할지를 정확하게 알 수 있지만 시스템의 전 영역에 패턴을 적용하지 않기 때문에 시스템 전체에 대한 안정성 및 신뢰성이 첫째 방법보다 낮아질 수 있다.

• 수집된 디자인 패턴 그룹핑(D300) - 수집된 디자인 패턴을 목표 시스템에 적용하기에 앞서 각 패턴의 목적을 정확하게 파악해야 한다. 수집된 패턴의 목적을 비교해서 컴포지션을 파악한다. 이를 위해 'Creational', 'Structural', 'Behavioral' 등의 컴포지션 분류를 기준으로 수집된 디자인 패턴을 그룹핑한다[1,11,16,17].

표 3 목적에 의한 디자인 패턴 구분

목적	디자인 패턴
Creational	Abstract Factory, Builder, Factory Method, Prototype, Singleton
Structural	Adapter, Bridge, Composite, Decorator, Façade, Proxy
Behavioral	Command, Interpreter, Iterator, Mediator, Memento, Observer, State, strategy, Visitor

이런 구분은 다음 단계인 패턴 매핑에 중요하고도 유용한 정보로 사용된다.

• 디자인 패턴 간의 관계 분석(D400) - 디자인 패턴 사이의 의존성 및 연관 관계를 파악한다. UML의 Use-Case간에 <<extend>>, <<include>>, generalize 등의 관계가 있는 것과 같이 디자인 패턴도 각각 독립적으로 사용되기 보다는 일련의 관계를 가지고 있어 여러 개의 디자인 패턴이 함께 사용된다[7,18-20].

표 4 패턴 사이의 관계

관계	표현	의미
X ← Y	X는 Y를 사용한다.	Use 또는 Composition
X → Y	X는 Y로 상세화된다.	Refine 또는Inheritance
X ⇔ Y	X는 Y와 대립한다.	Conflict

(* X, Y는 임의의 디자인 패턴)

X ← Y 관계는 X 패턴은 혼자 사용되지 않고 Y 패턴과 함께 사용되어야 하는 관계이다. X → Y 관계는 Y 패턴이 X 패턴을 상속 받는 관계이다. X ⇔ Y 관계는 X, Y 패턴이 각각 유사한 해법을 제시한다. 그러나 X, Y 두 패턴 중의 하나를 선택하면 나머지 하나의 패턴이 사용되어서는 않되는 관계이다.

D200에서 수집되고 D300에서 그룹핑된 디자인 패턴들을 대상으로 표 4를 기준으로 디자인 패턴 간의 관계를 식별한다.

- 작성된 시스템 아키텍처와 관련된 패턴 추출(D500) - 수집되어 그룹핑된 패턴들로부터 요구사항 분석 단계에서 추출된 기능적 요구사항, 비기능적 요구사항 및 아키텍처 선정 단계에서 선정된 아키텍처에 적합한 패턴을 추출한다. 이 때 적합한 패턴을 추출하기 위해 각 패턴의 문제 영역과 솔루션 영역을 비교한다.

3.4 패턴 매핑(P000)

- 시스템의 각 계층에 추출된 디자인 패턴 매핑(P100) - 선정된 시스템 아키텍처에 대해 시스템 계층화 단계에서 식별된 각 계층에 추출된 디자인 패턴을 매핑한다. 표 5는 효과적인 디자인 패턴 매핑을 위한 일반적 패턴 매핑을 보여준다.

표 5 계층별 디자인 패턴 매핑

계층	매핑된 디자인 패턴
프리젠테이션 계층	Intercepting Filter, Front Controller, View Helper, Composite View, Service To Worker, Dispatcher View
비즈니스 계층	Delegate, Value Object, Session Façade, Composite Entity, Value Object Assembler, Value List Handler, Service Locator
통합 계층	Data Access Object, Service Activator

- 최종 선정 디자인 패턴 관리(P200) - 각 시스템 계층별로 매핑된 디자인 패턴들이 최종적으로 패턴 기반 소프트웨어 개발에 사용되는 패턴이다. 이들 패턴들에 대해서는 문제영역, 솔루션 영역, 사용 사례 등을 별도의 패턴 명세서를 작성해서 관리한다.

4. 적용 사례 및 비교 평가

본 장에서는 제시한 프로세스를 적용하여 항공사의 국제화물 관련 항공업무 시스템을 개발해 나갔던 사례를 디자인 패턴 그룹화, 패턴 매핑 단계를 중심으로 보인다. 또한 본 논문에서 제시한 패턴 선정 프로세스와 관련 연구를 통해 제시한 기존 프로세스와의 차이점을 제시한다.

4.1 적용 사례

국내 항공사의 국제화물 관련 항공업무는 크게 미국 LA공항과 NEWYORK 공항의 화물터미널의 화물 반입, 적재, 반출, 수송 등의 업무로 구성된다. 이 시스템은 향후 전세계의 모든 자사 지점 공항의 화물터미널 운영시스템으로 적용되도록 공통업무, 특화업무 등을 파악하여 시스템 차원의 재사용을 궁극적인 목표로 하고 있다. 이 시스템은 IBM의 RS6000 머신 - DB 서버, WAS 서버 - 에 AIX UNIX를 운영체제로 탑재했고, Pentium 머신에 Windows 운영체제를 설치하여 웹 리포트 서버로 사용했다. 이들 메인 서버들은 국내에 위치하여 네트워크로 서비스를 제공하고, 미국 현지의 터미널에 있는 일반 컴퓨터와 PDA 장비를 이용해 항공 화물에 대한 수출입, 보관, 운송 등의 업무를 수행하도록 구축됐다.

- 요구사항 분석(R000)~아키텍처 선정(A000)

시스템 개발을 위한 요구사항 분석을 위해 WFD (Work Flow Diagram), UML의 Use Case 다이어그램을 이용해서 현행 업무를 분석하고, Activity 다이어그램과 Sequence 다이어그램을 이용해서 목표 시스템을 설계했다. 요구사항 분석과 현행 시스템의 분석 결과를 토대로 목표 시스템에 적합한 아키텍처를 선정하기 위해 전체 시스템 아키텍처, 페이지 아키텍처, 어플리케이션 아키텍처, 데이타베이스 아키텍처, 보안 아키텍처, 인터페이스 아키텍처, 네트워크 아키텍처 등 전체 7개의 분류 기준을 작성했다. 아키텍처 선정을 위한 각 분류별 세부 분류 요소는 표 6의 내용과 같다.

이런 검토 결과를 바탕으로 그림 4와 같은 시스템 아키텍처가 결정됐다.

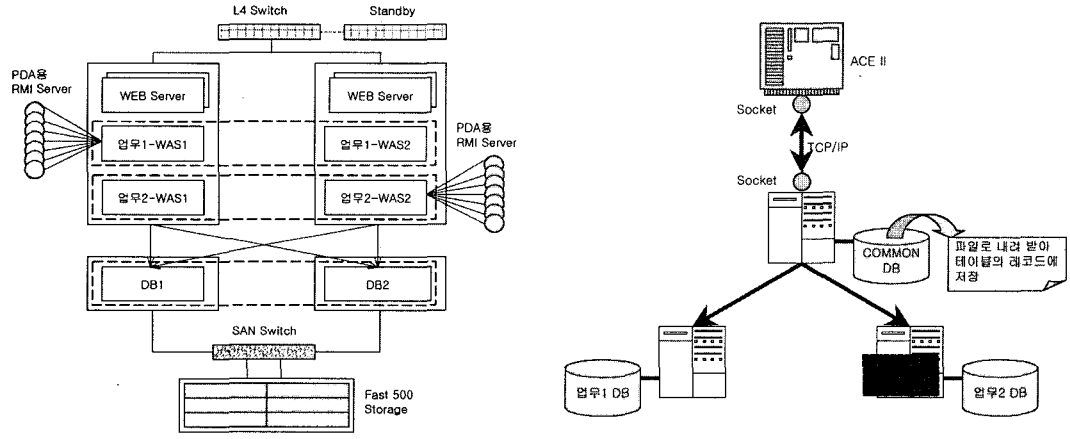
- 디자인 패턴 그룹화(D000)

선정된 아키텍처에 대해 적합한 디자인 패턴을 수집하기 위해 시스템의 계층구조 작성(D100)활동을 통해 표 7과 같이 클라이언트 계층, 프리젠테이션 계층, 비즈니스 계층, 통합 계층, 자원 계층으로 구분했다.

시스템을 계층화 한 후 각 계층에 적합한 디자인 패턴을 선정하기 위해 알려진 디자인 패턴 수집(D200) 활동을 수행하면서 GoF 디자인 패턴, J2EE 디자인 패턴 등을 분석했다. 이 과정에서 중요하게 고려된 것은 하드웨어/소프트웨어 아키텍처상의 문제였다. 모든 메인 서버(레거시 시스템 포함)는 국내에 있고, 터미널 내에서의 무선 통신을 이용해 데이타 교환, RMI를 통한 데이

표 6 아키텍처 선정을 위한 고려 요소

1단계 아키텍처 분류 기준	2단계 아키텍처 분류 기준	3단계 아키텍처 분류 기준
전체 시스템 아키텍처	시스템 아키텍처	- 현행 하드웨어 아키텍처 - 향후 하드웨어 아키텍처 - 향후 소프트웨어 아키텍처(물리적 구조) - 향후 소프트웨어 아키텍처 (논리적 구조)
	적용 기술	- 시스템 구성 기술 - 응용어플리케이션 구성 기술
	계층(Layer)별 적용 기술 및 사양	적용 기술 사양
페이지 아키텍처		
어플리케이션 아키텍처	- 어플리케이션 배치 구성도 - 계층별 소프트웨어 구성 및 배치 구성도 - 데이타베이스 연결 방법	
데이타베이스 아키텍처		
보안 아키텍처	- 화면 로그인 방법 - 권한 설정 방법	
인터페이스 아키텍처		
네트워크 아키텍처		



(a) 시스템 아키텍처 (b) 인터페이스 아키텍처

그림 4 터미널 운영시스템 아키텍처

표 7 시스템 계층 구조

계층	적용기술	용도
클라이언트 계층	Browser J2ME	Web Browser와 Mobile Browser를 이용해서 작업화면을 핸들링 PDA 단말기로부터 데이타를 입력 받거나, 업무 처리 결과를 PDA작업 화면에 출력
프리젠테이션 계층	HTTP JSP DHTML JavaScript	작업화면을 생성하고 사용자로부터 데이타 및 이벤트를 입력 받아 서비스 요청을 서버에 전달(특정 로직을 경유) 서버로부터 전달되는 응답결과를 화면에 출력(특정 로직을 경유)
비즈니스 계층	Servlet Java Bean RMI JSP Bean	화면에서 전달된 데이타 및 이벤트에 대해 데이타를 조작, 가공하고 업무 로직을 수행 입력, 수정, 삭제, 저장의 대상이 되는 비즈니스 데이타 가공 및 제어
통합 계층	TCP/IP Socket Demon	ACE, RAS 등과 인터페이스 데몬을 통해 통신하여 목표 데이타를 수집하고, 수집된 데이타를 특정 데이타 포맷에 맞는 토큰으로 파싱 후 데이타베이스에 저장
자원 계층	ISAM RPG	ACE: 개별 데이타 헤더를 갖고 있는ISAM 파일구조로 되어 있는 항공 운항 정보 데이타를 핸들링 하는 시스템 RAS: A400 기반에서 RPG 언어로 작성되어 있는 수입관리 시스템

타 전송 등은 미국 현지의 클라이언트와 국내의 메인 서버, 미국의 통관시스템 등을 분산환경으로 구성해야 한다는 것이었다. 이런 아키텍처 상의 특수성을 감안하여 표 7에서 언급한 5개의 계층을 중심으로 각 시스템 계층에 적용하기 위해 수집한 디자인 패턴들을 디자인 패턴 그룹핑(D300) 활동을 수행하여 표 8과 같이 그룹화 했다. 최종의 적용 대상 디자인 패턴을 추출하기 위해 표 8의 그룹화된 패턴들에 대해 작성된 시스템 아키텍처 및 구현 측면을 고려하였는데 이를 위해 적용된 요소들은 다음과 같다.

- 웹 브라우저에서 호출 URL이 보이지 않도록 할 것
- SQL과 비즈니스 로직을 소스 코드에서 분리 할 것
- 커맨드를 이용해서 비즈니스 기능을 분기시키고 화면을 이동시킬 것
- 대부분의 비즈니스 트랜잭션이 국내외를 오가며 발생하므로 네트워크 연결 횟수 및 트랜잭션 호출 횟수를 최소화 할 것
- 소스 코드의 재사용이 가능하도록 클래스를 추상화 할 것
- 목표 시스템이 전세계의 여러 지점에서 사용되어야 하므로 각 시스템 메시지에 대하여 국제화할 것
- 여러 개발자에 의해 비즈니스 로직이 개발되더라도 로직의 의미를 서로 쉽게 이해하고, 발생되는 문제에 대해 일관된 해결책을 제시할 수 있는 방법을 적용할 것

다음으로 디자인 패턴간의 관계 분석(D400) 활동을 수행하여 패턴 유형별로 추출된 디자인 패턴을 중심으로 X ← Y, X → Y, X ⇔ Y 관계를 고려하고, 이와 함께 추출된 패턴 각각에 결합도와 응집도를 분석했다. 즉 추출한 디자인 패턴 중에서 Factory Method 패턴은 반드시 Template Method 패턴 내에서 호출되어야 하므로 원래 사용 대상 디자인 패턴으로 추출되지 않은 Template Method와 같은 부가적인 패턴들도 사용해야 했다. 이 활동의 결과로 표 9와 같이 부가적으로 사용해야 하는 디자인 패턴을 추출했다.

표 8 추출된 디자인 패턴 그룹핑

패턴 유형	선정된 디자인 패턴	패턴 출처
생성 패턴	Abstract Factory	GoF
	Factory Method	GoF
	Singleton	GoF
구조 패턴	Adapter	GoF
	Bridge	GoF
	Composite View	J2EE
	Composite Entity	J2EE
	Façade	GoF
	Session Façade	J2EE
	Model-View-Controller	J2EE
	Business Delegate	J2EE
행위 패턴	Command	GoF
	Iterator	GoF
	State	GoF
	Visitor	GoF
	Front Controller	J2EE
	Fast Lane Reader	J2EE
	Data Access Object	J2EE
	Transfer Object	J2EE
	View Helper	J2EE

표 9 의존성 및 결합성을 고려해 선정된 부가 디자인 패턴

추출된 패턴 이름	부가 패턴	관계
Abstract Factory	Template	X ← Y
	Prototype	X → Y
[주1)]Façade	Mediator	X → Y
[주2)]Iterator	Memento	X ← Y
Singleton	Observer	X ← Y

주1) Façade 패턴은 Mediator 패턴에 의해 정제된다(Mediator → Façade)

주2) Singleton 패턴은 Observer 패턴에 의해 사용된다(Observer ← Singleton)

이 활동을 통해 추후 개발 단계에서 작성될 프로그램에 대해 코드의 중첩, 코드의 모호성, 코드의 복잡성, 코드 가독성 결여 등의 문제를 원천적으로 배제하여 효과적인 프로그램을 개발했다. 작성된 시스템 아키텍처와 관련된 패턴 추출(D500) 활동을 수행하여 시스템의 각 계층별로 생성 패턴 3개, 구조 패턴 8개, 행위 패턴 9개, 그리고 부가적으로 선택된 5개 등 총 25개의 패턴을 추출했다.

- 패턴 매핑(P000)

시스템의 각 계층에 추출된 디자인 패턴 매핑(P100) 활동을 수행하여 패턴의 유형별로 추출한 디자인 패턴을 시스템 계층화 단계에서 식별한 각 계층에 적절하게 매핑시켰다. 표 10은 목표 시스템을 위해 최종적으로 선정하여 계층에 매핑한 디자인 패턴 매팅 목록이다.

최종 선정 디자인 패턴 관리(P200) 활동을 통해 디자인 패턴을 사용하기 위한 명세서를 각각의 패턴 별로 작성하여 개발 단계에서 효과적으로 각 디자인 패턴을 활용 할 수 있도록지침을 제공했다.

4.2 비교 평가 및 효과

본 절에서는 제시된 패턴 선정 프로세스와 기존의 RUP 및 POAD와의 차이점을 보인다.

POAD는 컴포넌트와 같은 블록을 생성하여 소프트웨어를 개발하고자 할 때 설계 단계에서 컴포넌트를 효과적으로 설계하기 위한 접근법으로 주로 패턴 사이의 컴포지션 관계, 패턴의 인터페이스 사이의 관계를 중점적으로 고려해서 컴포넌트를 설계하도록 가이드하고 있다.

표 10 선정된 디자인 패턴 매핑

계층	선정된 패턴
Layer 1(Presentation)	Front Controller, Composite View, View Helper, Command
Layer 2(Control & Business Logic)	Business Delegate, Transfer Object, Façade, Session Façade, Composite Entity, Visitor, Model-View-Controller, Iterator, State
Layer 3(Data)	Fast Lane Reader, Abstract Factory, Factory Method, Singleton
Layer 4(Integration)	Adapter, Bridge, Data Access Object
Layer 5(Legacy System)	

(주) Layer 5는 이미 구축되어 있는 시스템이므로 구현을 위한 별도의 디자인 패턴을 고려하지 않았다.

표 11 제안한 패턴 선정 프로세스와 RUP, POAD의 비교

프로세스 / 비교항목	RUP	POAD	제안한 패턴 선정 프로세스
프로세스 자체에서 패턴 사용 지침 제공	×		○
시스템 아키텍처를 고려한 패턴 분류	×	△	○
패턴 적용 유도	개인	프로세스 자체	프로세스 자체
패턴 사용시 주된 고려 사항	개인의 지식 및 경험	• 패턴이 사용될 컴포넌트의 기능성 • 패턴의 컴포지션 • 패턴의 인터페이스	• 시스템의 계층(클라이언트, 프리젠테이션, 비즈니스, 통합, 자원) • 각 패턴의 목적(creational, structural, behavioral) • 패턴 사이의 결합도/응집도 • 패턴 사이의 관계(uses, refine, conflict)

그러나 POAD는 소프트웨어를 위해 생성할 컴포넌트의 기능성에 주로 초점을 맞춘후 그 기능성을 만족할 패턴을 선정하므로 전체 소프트웨어 차원의 하드웨어 아키텍처나 소프트웨어 아키텍처를 고려해서 패턴을 선정하는 아키텍처 중심의 설계 패턴분석이 부족하다. 또한 패턴과 패턴 인터페이스의 관계만을 고려함으로써 해당 패턴이 소프트에어의 비즈니스 로직이나 제어를 위해 사용될 패턴인지, 데이타나 레거시 시스템과의 연동을 위해 사용될 패턴 인가 등에 대한 패턴 분류가 미흡하다. 이런 POAD의 몇몇 미흡한 부분을 위해 본 논문에서는 아키텍처 선정(A000) 단계, 디자인 패턴 그룹화(D000) 단계의 시스템의 계층구조 작성(D100) 활동과 패턴 매핑(P000) 단계의 시스템의 각 계층에 추출된 디자인 패턴 매핑(P100) 활동을 수행하도록 제시했다. 패턴을 이용해서 컴포넌트를 개발하거나 응용 소프트웨어를 개발할 때 기존에는 개인의 지식이나 경험에 의존해서 패턴을 적용함으로써 구조적으로 완전하지 못한 패턴 기반 소프트웨어가 개발되었지만, 제안한 효과적인 패턴 선정프로세스를 통해 응용 소프트웨어에 패턴을 적용함으로써 보다 정확하고 완전한 패턴 기반 소프트웨어를 개발할 수 있고, 이 소프트웨어는 추후 보다 원활하게 리팩토링 및 재공학을 효과적으로 수행할 수 있다[7,21,22].

5. 결론 및 향후 연구 과제

디자인 패턴이 여러 문제 영역에 대한 공통된 해결책을 제시하는 경험적 산물로서 분석, 설계, 개발 과정에서 결과물의 명확성, 의사 소통의 원활함, 진화 가능한 소프트웨어로의 발전 등을 제공함을 알 수 있다. 그럼에도 불구하고 현재까지 많은 소프트웨어 개발 프로젝트에서 디자인 패턴이 적절하고 빈번하게 사용되지 못했음도 알고 있다. 이에 본 논문에서는 보다 더 효과적이고 효율적으로 패턴을 선정해서 패턴 기반 소프트웨어를 개발할 수 있도록 '요구사항 분석→아키텍처 선정→디자인 패턴 그룹화→패턴 매핑'의 패턴 선정 프로세스를 제시했다.

적절한 패턴을 선정하기 위한 중요한 활동은 요구사항 분석 단계의 목표 시스템 분석/설계 활동과 적용할 대상 기술 선정 활동이고, 이 활동의 결과를 토대로 아키텍처 선정 단계에서 개발할 소프트웨어에 대한 적절한 하드웨어, 소프트웨어 아키텍처가 선정되어야 함을 보였고, 각 디자인 패턴마다 특성이 있어서 함께 묶여서 사용될 수 있고, 패턴 간에 관계가 있어 선정 및 사용시에 패턴의 결합도와 응집도가 심도 있게 고려되어야 함을 보였다. 또한 제시한 프로세스를 특정 항공사의 항공관련 업무에 적용하여 전 세계에서 운영되는 대규모의 소프트웨어 개발 과정에 패턴을 적절하게 적용함으로써 안정적이고, 재사용 및 확장을 위한 진화 가능한 소프트웨어를 효과적으로 개발 가능함을 보였다.

RUP 또는 MaRMI III 등의 프로세스 또는 방법론에서는 디자인 패턴을 설계단계에서 고려 또는 적용하도록 가이드하고 있지만 구체적인 디자인 패턴 선정 방법 및 절차는 언급되어 있지 않다. 그러므로 향후에는 본

논문에서 제시한 패턴 선정 프로세스를 RUP 또는 MaRMI III 등에 플러그-인 플러그 아웃(Plug-In Plug-Out)해서 사용할 수 있는 방법에 대한 연구가 시행될 것이고, 디자인 패턴을 적용하여 개발된 소프트웨어의 품질은 어떻게 평가할 것인가에 대한 연구가 수반될 것이다.

참 고 문 헌

[1] E. Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[2] Deepak Alur, John Crupi and Dan Malks, *Core J2EE Patterns: Best Practices and Design Strategies*, Prentice Hall, 2001.

[3] Christopher Alexander, Sara Ishikawa and Murray Silverstein, *A Pattern Language: Towns, Buildings, Construction*, Oxford University press, 1977.

[4] Grant Larsen, Designing component-based frameworks using patterns in the UML, *Communications of the ACM*, Volume 42 Issue 10, pp:38-45, October, 1999.

[5] James O. Coplien, Douglas C. Schmidt, *Pattern Languages of Program Design*, Addison Wesley, 1995.

[6] Tiffany Winn, Paul Calder, Is This a Pattern?, *IEEE Software* n. 1, pp:59-65, January/February, 2002.

[7] Ladan Tahvildari, Kostas Kontogiannis, On the Role of Design Patterns in Quality-Driven Re-engineering, *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering (CSMR'02), IEEE*, pp:230-240, March 11-13, 2002.

[8] Philippe Kruchten, *The Rational Unified Process An Introduction (2nd Edition)*, Addison Wesley, 2000.

[9] Craig Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process (2nd Edition)*, Prentice Hall, 2002.

[10] Rational Unified Process Evaluation (Rational Unified Process Evaluation Assembly V2003.06.00), Available from: http://www-306.ibm.com/software/awdtools/rup/

[11] Sherif M. Yacoub, Hany H. Ammar, Pattern-Oriented Analysis and Design (POAD): A Structural Composition Approach to Glue Design Patterns, *Technology of Object-Oriented Languages and Systems (TOOLS 34'00), IEEE*, pp:273-282, July 30-August 4, 2000.

[12] Sherif M. Yacoub, Hany H. Ammar, *Pattern-Oriented Analysis and Design: Composing Patterns to Design Software Systems*, Addison Wesley, 2003.

[13] Rudolf K. Keller, Reinhard Schauer, Design Components: Towards Software Composition at the Design Level. *Proceedings of 20th International Conference on Software Engineering, ICSE'98*, pp:302-311, April 19-25, 1998.

[14] Len Bass, Paul Clements and Rick Kazman, *Software Architecture in Practice*, Addison-Wesley, 2003.

[15] Philippe Kruchten, The 4+1 View Model of Architecture, *IEEE Software*, (Vol. 12, No. 6), pp:42-50, November, 1995.

[16] Category Pattern, Available from: http://c2.com/cgi/wiki?CategoryPattern

[17] Sherif M. Yacoub, Hengyi Xue, Hany H. Ammar, POD: A Composition Environment for Pattern-Oriented Design, *Technology of Object-Oriented Languages and Systems (TOOLS 34'00), IEEE*, pp:263-272, July 30-August 4, 2000.

[18] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, *Pattern-Oriented Software Architecture, A System of Patterns*, John Wiley & Sons, 1996.

[19] Walter Zimmer, Relationships between design patterns. In *Pattern Languages of Program Design*. Addison-Wesley, 1994.

[20] William B. McNatt, James M. Bieman, Coupling of Design Patterns: Common Practices and Their Benefits, *Proceedings of the 25th International Computer Software and Applications Conference on Invigorating Software Development (COMPSAC '01)*, pp:574-579, October 08-12, 2001.

[21] Carmen Zannier, Tool support for refactoring to design patterns, *Companion of the 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA'02, pp:122-123, November 4-8, 2002.

[22] Taichi Muraki, Motoshi Saeki,Metrics for applying GOF design patterns in refactoring processes, *Proceedings of the 4th International Workshop on Principles of Software Evolution, IWPSE'01*, pp:27-36, September 10-11, 2001.

최 진 명

1998년 서울산업대학교 전자계산학과 공학사. 2000년 숭실대학교 컴퓨터학과 공학석사. 2001년~현재 숭실대학교 컴퓨터학과 박사과정. 관심분야는 디자인 패턴, 컴포넌트 기반 소프트웨어 재사용, 소프트웨어 프로세스, 소프트웨어 개발 방법론

류 성 열

1997년 2월 아주대학교 컴퓨터학부(공학박사). 1997년 3월~1998년 3월 George Mason University 교환교수. 1981년 3월~현재 숭실대학교 정보과학대학 컴퓨터학부 교수. 1998년 3월~2001년 2월 숭실대학교 정보과학대학원 원장. 1998년 3월~2004년 7월 숭실대학교 전자계산원 원장. 관심분야는 리엔지니어링, 소프트웨어 유지보수, 소프트웨어 재사용, 소프트웨어 재공학/역공학, 소프트웨어 테스팅