아키텍처 기반 소프트웨어 개발을 지원하는 효과적인 소프트웨어 아키텍처 평가 방법

(An Approach to Effective Software Architecture Evaluation in Architecture-Based Software Development)

최 희 석 † 염 근 혁 ††

(Heeseok Choi) (Keunhyuk Yeom)

요 약 소프트웨어 아키텍처는 소프트웨어 개발에 참여하는 사람들간의 원활한 의사 소통과 시스템 설계 결정에 대한 합리적 판단을 가능하게 하는 상위 수준의 시스템 추상화이다. 이러한 소프트웨어 아키텍처에 대한 평가는 소프트웨어에 요구되는 품질을 소프트웨어 개발 전에 미리 예측하거나, 고품질 소프트웨어를 개발하는데 적합한 아키텍처의 선택 및 향상을 가능하게 한다. 그러나 현재의 아키텍처 평가 방법은 아키텍처 평가의 초기 입력물에 대한 정의가 미흡하고, 평가 과정이 주관적인 프로세스에 많이 의존하거나 혹은 체계적인 프로세스를 정의하고 있지 않다. 뿐만 아니라 아키텍처 평가 정보들의 표현에 대한 고려가 부족하다.

본 논문에서 제안하는 방법은 아키텍처 평가의 주요 입력물인 요구사항을 기능적 요구사항과 비기능적 요구사항으로 분리하여 다루고, 소프트웨어 아키텍처는 UML을 이용한 "4+1" 뷰 아키텍처 모델을 바탕으로 명확하게 정의한다. 이를 통하여 아키텍처 평가의 목표 및 평가 범위, 평가 대상을 분명하게 결정한다. 그리고 아키텍처 평가의 중요 정보들인 부분 설계, 설계 결정, 근거 데이타, 품질 등의 정보를 단계적으로 결정하기 위한 체계적이고 객관적인 프로세스를 제시한다. 또한 평가 결과에 있어서는 아키텍처 평가 과정에서 결정된 정보들을 구조화된 형태로 나타냄으로써 품질 예측 및 아키텍처의 향상과 선택이 가능하도록 돕는다.

키워드 : 소프트웨어 아키텍처 평가, 4+1 뷰 모델, 아키텍처 기반 소프트웨어 개발

Abstract Software architecture representing a common high-level abstraction of a system can be used as a basis for creating mutual understanding among all stakeholders of the system. In determining a software architecture's fitness with respect to its desired qualities as well as in improving a software architecture, software architecture evaluation is importantly performed. However most of architecture evaluation methods are not still sufficient in that they do not explicitly consider artifacts discussed during architecture evaluation and their processes are not systematic. As a result, we are hard to follow them.

To address these problems, this paper presents the method to evaluate systematically a software architecture with respect to its desired qualities. In this approach, the functional and non-functional requirements are separately handled, and software architecture is represented in the 4+1 view model using UML. Through this initial consideration, the important artifacts such as goals, scope, and target of evaluation are clearly determined. Also, the method provides the well defined process to produce the important evaluation artifacts such as sub-designs, design decisions, rationale, qualities from inputs. In addition, it enables us to determine satisfaction of a architecture with respect its desired qualities or improve a architecture through the structured evaluation results.

Key words : Software architecture evaluation, 4+1 view model, Architecture-based development

· 본 연구는 한국과학재단 목적기초연구(R02-2000-00276)지원으로 수행되었음.

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

논문접수 : 2001년 8월 20일
심사완료 : 2002년 2월 25일

1. 서 론

점차 복잡하고 다양해지는 소프트웨어의 품질 향상과 시기 적절한 소프트웨어 생산, 요구사항 변화에 대한 효과적 대응을 위하여, 소프트웨어를 재사용 가능한 컴포넌트 형태로 개발하는 기술이 빠르게 발전하고 있다. 컴

포넌트 기반 소프트웨어 개발에 있어서 시스템에 대한 상위 수준의 추상화를 나타내는 소프트웨어 아키텍처는 소프트웨어 개발에 참여하는 사람들간의 원활한 의사 소통과 시스템 설계 결정에 대한 합리적 판단을 가능하게 한다[1,2]. 또한 소프트웨어 개발에 참여하는 사람들이 모두 인지할 수 있는 형태로 시스템 구조와 그 시스템을 구성하는 컴포넌트들의 동작을 보여줄 수 있으므로 고품질 소프트웨어를 개발하는 데 있어서 필수적이라고 할 수 있다[2,3]. 특히 대규모의 소프트웨어 시스템에 요구되는 품질의 달성은 구현 언어의 선택, 상세 설계, 알고리즘, 데이타 구조, 테스팅 등과 같은 코드 수준의 활동에 의해 결정되기 보다는 초기 단계의 중요한 설계 결정을 포함하고 있는 소프트웨어 아키텍처에 크게 의존한다[4]. 따라서 소프트웨어를 개발하기 전에 소프트웨어 아키텍처에 대한 평가를 통하여 개발될 소프트웨어가 얼마나 잘 품질 요구를 만족시킬 수 있는가를 결정하는 것이 필요하다. 이러한 평가 결정은 소프트웨어 품질 달성에 적합한 소프트웨어 아키텍처를 선택하거나 제시된 아키텍처를 향상시킬 수 있도록 하기 때문에, 궁극적으로 개발될 소프트웨어의 품질 향상에 기여할 수 있다는 점에서 특히 중요하다. 그러므로 소프트웨어 아키텍처가 요구 문서에서 기술하는 품질 요구를 충족시킬 수 있는가를 예측하기 위한 체계적인 평가 방법이 필요하다.

소프트웨어 아키텍처 평가에 관한 대표적인 연구로는 시나리오 기반의 평가 방법[4,5,6], 아키텍처 기술 언어(Architecture Description Languages, ADLs)를 이용한 평가 방법[7,8], 그리고 시뮬레이션 및 프로토타이핑 등을 이용한 실험적 평가 방법[9,10] 등이 있다. 시나리오 기반의 평가 방법은 품질 요구사항의 실질적인 의미를 구체화하는 일련의 시나리오 집합을 개발하여 아키텍처를 평가하는 방법이다. 아키텍처 기술 언어를 이용한 평가 방법은 비정형화된 아키텍처가 제공해주지 못하는 많은 정보를 정형화하여 아키텍처 구성 요소에 명확하게 나타냄으로써 아키텍처 평가를 지원하는 방법이다. 그리고 시뮬레이션 및 프로토타이핑 등을 이용한 실험적 평가 방법은 아키텍처를 구성하는 주요 컴포넌트에 대한 부분적인 구현과 실행을 통하여 아키텍처를 평가하는 방법이다.

그러나 이러한 방법들은 평가의 목표 및 평가 범위, 평가 대상에 대한 정의가 불명확하다[9,11]. 그리고 평가 과정이 모호하여 점진적이고 반복적인 평가가 어렵고, 평가 결과물에 대한 명시적 표현이 부족하다. 또한 다양한 시스템 관련자들의 관심을 효과적으로 분리하여 나타내줄 수 있는 다중 뷰(Views) 모델에 대한 고려가 부족하다.

본 논문에서 제안하는 방법은 아키텍처 평가의 초기 입력물인 소프트웨어 아키텍처와 요구사항을 명확하게 다룸으로써, 평가 목표 및 평가 범위, 그리고 평가 대상을 분명하게 결정하고 나타낸다. 그리고 이를 바탕으로 평가 수행에 대한 체계적이고 객관적인 프로세스를 제시한다. 또한 평가 결과에 있어서는 구조화된 형태의 결과물을 제시함으로써 아키텍처를 통한 품질 예측에 중요한 평가 정보들을 잘 나타낸다. 그러므로 궁극적으로 품질 예측 및 아키텍처의 향상과 선택이 가능하도록 돕는다.

2. 관련 연구

소프트웨어 아키텍처 평가는 제시된 소프트웨어 아키텍처가 개발될 소프트웨어에 대하여 요구되는 품질 특성을 충족시킬 수 있는가를 아키텍처 수준에서 평가하는 것이다[9,12]. 소프트웨어 아키텍처 평가에 있어서 주요 입력물은 크게 소프트웨어 아키텍처와 소프트웨어 요구사항이다. 그러므로 소프트웨어 아키텍처와 소프트웨어 요구사항에 대하여 먼저 살펴본다. 다음으로 소프트웨어 아키텍처 평가의 의미를 정확히 이해하고, 기존의 아키텍처 평가를 위한 주요 방법들에 대하여 기술한다.

2.1 소프트웨어 아키텍처

소프트웨어 아키텍처는 컴포넌트들의 구조, 상호 관계, 설계 및 변경을 가이드하는 원칙 등을 나타내는 시스템에 대한 상위 수준의 추상화이다[13,14]. 소프트웨어 아키텍처는 다음과 같은 이점을 제공한다[1,2].

첫째, 상호 의사소통을 원활하게 함

소프트웨어 아키텍처는 대부분의 시스템 관련자(Stakeholders)간에 상호 이해와 의사 소통을 위한 기초로 사용될 수 있다.

둘째, 초기의 설계 결정을 포함

소프트웨어 아키텍처는 초기 단계의 시스템 설계 결정에 대하여 구체적 근거 자료를 제공한다.

셋째, 시스템의 추상화를 제공

소프트웨어 아키텍처는 시스템이 어떻게 구조화되고, 그것의 컴포넌트들이 어떻게 함께 동작할 수 있는가에 대한 상위 수준의 모델을 제공한다.

소프트웨어 아키텍처를 표현하기 위한 대표적인 모델로서 Perry & Wolf의 모델[15], Shaw & Garlan의 모델[16], 그리고 Kruchten의 "4+1" 뷰 모델[3] 등이 소개되어 있다. 표 1은 이들 세 가지 모델들의 특성을 요약하여 나타낸 것이다.

표 1 대표적인 소프트웨어 아키텍처 모델들

[illegible] 아키텍처 모델	
요소	의미
Elements	처리 요소, 데이타 요소, 연결 요소
Form	가중치가 부여된 성질, 관계
Rationale	아키텍처 설계의 결정들에 작용한 요인들
[illegible] 아키텍처 모델	
요소	의미
Components	할당된 기능을 수행하는 계산 요소
Connectors	컴포넌트들의 상호작용을 중재
Patterns	컴포넌트들과 커넥너들의 합성 패턴에 대한 제약 조건들
[illegible] 아키텍처 모델 (4+1 뷰 모델)	
관점	의미
Use case view	시스템 시나리오들의 집합
Logical view	주로 기능적 요구사항들을 반영한 모델
Process view	동시성과 동기화 측면을 반영한 모델
Development view	개발 환경에서의 소프트웨어 정적 구조를 기술한 모델
Deployment view	소프트웨어의 하드웨어로의 배치 관계를 표현한 모델

표 1에서 나타낸 아키텍처 모델 중에서 "4+1" 뷰 모델은 그 아키텍처와 관련된 다양한 사람들(사용자, 개발자, 시스템 공학자, 프로젝트 관리자 등)의 관심을 효과적으로 분리시킬 수 있다. 또한 기능적 요구사항과 비기능적 요구사항을 분리하여 다룰 수 있도록 한다. 5 가지 뷰들 각각은 그 뷰가 포함하여야 하는 정보를 전달하기 위하여 다양한 표기법을 이용하여 표현될 수 있다. 대표적인 표기법으로 표준 모델링 언어인 Unified Modeling Language(UML)가 많이 이용된다. 최근에는 다양한 도메인의 응용 시스템 개발에 UML을 이용한 "4+1" 뷰 아키텍처 모델이 폭넓게 활용되고 있다[17,18,19].

2.2 소프트웨어 요구사항

소프트웨어 요구사항은 개발되는 소프트웨어에 대한 이해를 전달하는 내용을 담고 있다. 이러한 요구사항은 크게 두 가지로 분류할 수 있다[20]. 하나는 기능적 요구사항으로 시스템이 수행하는 기능성을 나타내고, 다른 하나는 비기능적 요구사항으로 기능성을 달성하는 데 있어서 부가되는 제약 조건 및 품질 특성을 나타낸다. 일반적으로 비기능적 요구사항은 하나 또는 그 이상의 기능적 요구사항과 관련되어 있다[20]. 그리고 비기능적 요구사항에 의해 주로 결정되는 품질 특성 역시 크게 두 가지로 분류될 수 있다[21]. 하나는 사용자 관점에서 시스템의 사용과 관련된 품질 특성으로 유용성, 효율성, 유연성, 무결성, 신뢰성 등이 있다. 다른 하나는 개발자 및 유지보수자 관점에서 개발 및 유지보수 활동과 관련된 품질 특성으로 유지보수성, 이식성, 재사용성 등이 있다.

2.3 소프트웨어 아키텍처 평가에 관한 기존 연구들

소프트웨어 아키텍처 평가는 제시된 소프트웨어 아키텍처가 개발될 소프트웨어에 대하여 요구되는 품질 특성을 충족시킬 수 있는가를 아키텍처 수준에서 평가하는 것이다[9,12]. 아키텍처 수준에서의 평가는 다음과 같은 점에서 중요하다.

- 대규모의 소프트웨어 시스템에 요구되는 품질의 달성은 구현 언어의 선택, 상세 설계, 알고리즘, 데이타 구조, 테스팅 등과 같은 코드 수준의 활동에 의해 결정되기 보다는 시스템 설계 초기 단계의 중요한 설계 결정을 포함하고 있는 소프트웨어 아키텍처에 크게 의존한다[4].
- 완성된 시스템으로 실체화되기 전에 아키텍처를 평가함으로써 배포될 시스템의 품질 목표를 만족시킬 수 없도록 할 가능성이 있는 위험 요소를 크게 줄일 수 있다. 특히, 아키텍처 설계 결정에 의한 위험 요소는 소프트웨어가 개발된 이후에 변경하는 것이 거의 불가능하다[22].
- 아키텍처 평가 결정은 소프트웨어 품질 달성에 적합한 소프트웨어 아키텍처를 선택하거나 제시된 아키텍처를 향상시킬 수 있도록 하기 때문에, 궁극적으로 개발될 소프트웨어의 품질 향상에 기여할 수 있다[6,23].

소프트웨어 아키텍처를 평가하기 위한 대표적인 방법으로는 시나리오 기반의 평가 방법[4,5,6], 아키텍처 기술 언어(ADLs)를 이용한 평가 방법[7,8], 그리고 시뮬레이션 및 프로토타이핑 등을 이용한 실험적 평가 방법[9,10] 등이 있다.

2.3.1 시나리오 기반의 아키텍처 평가

대부분의 소프트웨어 품질은 복잡하고 모호하다[12]. 이에 대해 시나리오는 시스템 사용 환경을 고려하여 모호한 품질 특성에 대한 실질적인 의미를 보다 구체적으로 제공해 줄 수 있다는 측면에서 중요하게 다루어진다[5]. 시나리오 기반의 아키텍처 평가는 이와 같이 품질 요구사항의 실질적인 의미를 구체화하는 일련의 시나리오 집합을 개발하여 아키텍처를 평가하는 방법이다[4,5,12]. 일련의 시나리오 집합은 소프트웨어 시스템과 관련된 다양한 사람들과의 대화와 상호 이해, 시스템 사용 측면에 대한 고려 등을 통하여 개발된다. 개발된 시나리오에 따라 제시된 아키텍처를 정제하거나 새롭게

기술하고, 시나리오 집합을 아키텍처에 적용하여 평가한다. 이를 통하여 아키텍처가 충족시켜야 할 품질을 예측하고, 잠재적 위험 요소를 결정한다. 시나리오를 기반으로 한 대표적인 평가 방법으로는 SAAM[5], ATAM[4] 방법 등이 있다.

그러나 시나리오 기반의 평가 방법은 다음과 같은 문제점을 가지고 있다.

- 적절한 시나리오 집합을 결정하는 것이 어렵다[6].
- 평가 과정의 진행이 주관성에 의해 많이 좌우된다. 그러므로 평가의 반복성에 문제가 있다[5,12].
- 평가 입력물에 대한 명확한 정의가 없으므로 평가 목표 및 평가 범위, 평가 대상이 불분명하다. 그리고 품질 예측 및 아키텍처 선택과 향상에 중요한 영향을 주는 평가 정보들에 대한 명시적인 표현이 부족하다.

2.3.2 아키텍처 기술 언어를 이용한 아키텍처 평가

아키텍처 기술 언어(ADLs)는 구현 수준의 컴포넌트 관계보다 높은 추상화 수준의 상호 관계 및 시스템 구조를 정형명세 하는 언어이다[7,8]. 아키텍처 기술 언어는 특정한 종류의 시스템에 요구되는 품질 특성을 잘 반영하여 충분한 아키텍처 정보를 제공할 수 있도록 다양한 종류가 있다. 아키텍처 기술 언어를 이용한 아키텍처 평가는 비정형화된 아키텍처가 제공해주지 못하는 많은 정보에 대하여 필요한 아키텍처 구성 요소를 결정하고, 그것들에 대한 정보를 정형화하여 제공함으로써 아키텍처 평가를 지원하는 방법이다.

그러나 이러한 접근 방법 역시 다음과 같은 문제점을 가지고 있다.

- 아키텍처 전체의 행동 방식보다는 아키텍처를 구성하는 각 요소의 행동 양식에 중점을 둔다[24].
- 평가하려는 아키텍처를 기술하는 ADL의 표현 능력에 의해 품질 특성에 대한 아키텍처 평가가 제한적이다. 이것은 ADL을 이용한 평가가 컴포넌트, 커넥터에 대한 아키텍처 수준의 정보에 의존적이기 때문이다[24].
- 아키텍처 평가를 위한 체계적인 프로세스를 정의하고 있지 않다.

2.3.3 실험적인 아키텍처 평가

시뮬레이션 또는 프로토타이핑 등을 이용하여 아키텍처를 실험적으로 평가하는 것은 그 아키텍처의 주요 컴포넌트에 대한 부분적인 구현을 필요로 하는 방법이다[10]. 이 방법은 아키텍처 구성 요소들의 행동 방식과 요소들 사이의 상호작용을 매우 정확하게 기술하도록 함으로써, 설계 초기에 설계상의 불일치 문제를 드러낼 수 있다[9]. 또한 시뮬레이션 모델을 실제로 실행하여 봄으로써 특정 품질 특성을 평가할 수 있다[9]. 그러나 이 방법은 아키텍처에 대한 부분적인 구현 및 별도의 시뮬레이션 모델 등을 개발해야 하므로, 상위 수준에서의 아키텍처 평가를 위해서는 적합하지 못하다.

3. 소프트웨어 아키텍처 평가 방법

본 논문에서 제시하는 소프트웨어 아키텍처 평가 방법은 표 2에서 나타낸 바와 같이 4 가지 범주[9]에서 각각 'General', 'Medium', 'Middle', 그리고 'Artifact'이라는 성격을 가진다.

표 2에서 알 수 있듯이, 제안하는 방법은 다양한 도메인에 존재할 수 있는 여러 특성들을 다루고, 표현되는 아키텍처 정보의 수준은 UML을 이용한 "4+1" 뷰 모델에서 표현될 수 있는 정보에 의해 결정된다. 그리고 아키텍처 평가는 아키텍처만이 존재하는 초기 단계에서 이루어지며, 평가 대상은 소프트웨어 아키텍처가 된다.

다음으로 평가 대상인 소프트웨어 아키텍처가 포함하는 아키텍처 수준의 정보는 표 3과 같이 정리될 수 있다.

표 2 제안한 아키텍처 평가 방법의 성격

범주	내용	구분	비고
Generality	Focus on issues	General	일반적인 이슈에 초점을 두고, 어떠한 도메인의 아키텍처에 적용될 수 있음
		Domain-specific	주어진 도메인의 특정 이슈와 관련됨
		System-specific	주어진 시스템의 특정 이슈와 관련됨
Level of detail	How much information about the architecture	Coarse	상세 정보를 필요로 하지 않음
		Medium	UML로 표현되는 4+1 뷰 아키텍처 모델
		Fine	보다 상세한 정보를 필요로 함
Phase	Three phases of interest to architecture evaluation	Early	아키텍처와 시스템 모두 존재 안함
		Middle	아키텍처만 존재
		Post	아키텍처와 시스템 모두 존재함
What is Evaluated	kinds of questions that could be answered	Artifact	아키텍처
		Process	아키텍처와 아키텍체 개발 과정

표 3 평가 대상인 아키텍처가 포함하는 정보

뷰	아키텍처 구문에 포함
Use case view	-시스템의 주요 목적 -주요 목적 수행과 관련된 일련의 행동 흐름
Logical view	-설계에 대한 개념적 모델 -정적 구조와 동적인 상호작용
Development view	-개발 환경에서의 소프트웨어 정적 구조 -구현 모듈과 그것들간의 상호관계 -컴포넌트의 그룹화 및 분리, 가시적 인터페이스 정의
Process view	-동시성 및 동기화 -자원의 사용, 병렬 수행, 비동기적 이벤트의 처리 -작업 그룹과 복제단위로서의 프로세스 분할 -작업 단위간의 상호작용 메커니즘 -메시지 흐름 및 프로세스의 부하
Deployment view	-소프트웨어 구성 요소의 하드웨어로의 배치 관계 -Logical, Process, Development 뷰에서 결정된 요소들의 처리 노드로의 매핑 관계

본 논문에서 제시하는 소프트웨어 아키텍처 평가 방법은 그림 1에서 나타낸 바와 같이 크게 평가 준비, 평가 수행, 평가 완료 단계로 구성된다. 평가 준비 단계에서는 아키텍처 평가의 초기 입력물인 소프트웨어 아키텍처와 요구사항을 명확하게 정의한다. 평가 수행 단계에서는 아키텍처에 요구되는 품질 특성에 관련된 아키텍처 설계 결정 및 각 설계 결정에 대한 근거 데이타(Rationale)를 추출하고 결정한다. 마지막으로 평가 완료 단계에서는 품질 예측에 사용될 데이타를 정의하고, 소프트웨어 아키텍처의 선택 및 향상에 활용될 수 있는 평가 데이타를 구조화하여 표현한다.

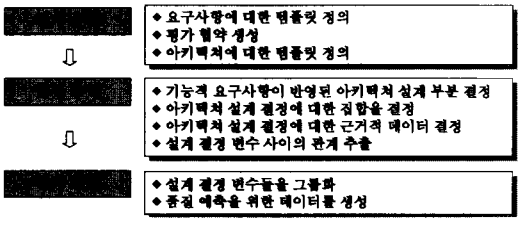

그림 1 제시하는 아키텍처 평가 방법의 절차

3.1 평가 준비

평가 준비 단계는 아키텍처 평가에 있어서 결정되어야 할 아키텍처 평가의 목표 및 평가 범위, 평가 대상을 명확하게 정의하는 단계이다. 이를 위하여 이 단계에서는 요구사항과 아키텍처에 대한 템플릿(Template)을 정의하고, 평가 목표와 평가 범위에 대한 평가 협약(Contract)을 결정한다. 평가를 위한 템플릿과 평가 협약은 전체 아키텍처 평가 과정을 이끌어 나가는 역할을 한다.

3.1.1 요구사항에 대한 템플릿 정의

소프트웨어 요구사항은 다양한 형태로 존재 가능하다. 특히 기능적 요구사항과 비기능적 요구사항이 분리되어 표현되거나 혼합되어 표현될 수 있다. 그러나 소프트웨어에 대한 기능적 요구사항과 비기능적 요구사항의 혼합은 평가 목표 및 평가 범위 결정을 어렵게 한다. 이와 함께 아키텍처 평가에 있어서 중요하게 다루어질 품질 특성에 대한 결정을 어렵게 한다. 따라서 요구사항에 대한 템플릿은 요구사항에 대하여 일정한 형태를 정의한다. 그러한 템플릿을 정의하기 위한 과정은 다음과 같다.

① 기능적 요구사항과 비기능적 요구사항을 분리해낸다. 기능적 요구사항은 시스템이 제공하는 기능성을 나타내고, 비기능적 요구사항은 기능성을 달성하는 데 있어서 부가되는 제약조건 및 품질 특성으로서 일반적으로 주어진 기능을 '얼마나 잘' 달성할 것인가와 관련된다.

② 분리된 기능적 요구사항과 비기능적 요구사항간의 관계를 형성한다. 비기능적 요구사항은 하나 또는 그 이상의 기능적 요구사항과 관련된다.

3.1.2 평가 협약 생성

평가 협약은 평가 목표 및 평가 범위를 정의하고 있으며, 제시된 소프트웨어 아키텍처에 대하여 요구되는 품질 특성을 포함하고 있다. 평가 협약은 다음과 같은 과정을 통하여 결정한다.

① 아키텍처 평가의 범위를 결정한다.

▪ 요구사항에 대한 템플릿에서 분리 정의된 기능적 요구사항을 표 4에서 나타낸 기능 분류에 따라 'Primary' 기능, 'Mandatory' 기능, 그리고 'Optional' 기능으로 분류한다. 이때 'Primary'로 분류된 기능 중 일부는 다른 'Primary' 기능의 'Mandatory' 기능이 될 수 있다.

이것은 평가 범위를 결정하는 데 있어서 어떠한 'Primary' 기능이 선택되었을 때, 그것의 'Mandatory'로 분류된 기능이 동시에 'Primary' 기능으로 존재할 때 그 'Primary' 기능도 평가 범위에 포함됨을 의미한다. 표 4는 기능적 요구사항의 기능적 특성에 따른 분류 항목을 나타낸 것이다.

표 4 기능적 특성에 따른 분류

분류	의미
Primary	시스템의 주요 목적을 나타내는 기능
Mandatory	주요 목적과 관련하여 있어야 하는 기능
Optional	주요 목적과 관련하여 선택적으로 존재할 수 있는 기능

• 분류된 기능적 요구사항들은 표 5에서 나타낸 기능간의 관계에 따라 연결된다. 이것은 평가 범위 결정시 'Primary' 기능에 대하여 평가 범위를 결정하면 그것과 관련된 'Mandatory' 기능과 'Optional' 기능이 평가 범위에 포함되도록 하는 데 활용된다.

표 5 분류된 기능들간의 관계

분류	내용
supported-by	Primary 기능과 Mandatory 기능 사이의 관계
associated-with	Primary 기능과 Optional 기능 사이의 관계

• 다음으로 개발될 소프트웨어에 대한 관점(viewpoints)을 결정한다. 소프트웨어 시스템은 크게 두 가지 관점으로 나누어 생각할 수 있다[12]. 하나는 시스템의 사용자 관점이고, 다른 하나는 개발자 또는 유지관리자 관점이다. 소프트웨어에 대한 관점에 따라 평가 대상인 아키텍처에 요구되는 품질 특성에 대한 결정에 영향을 줄 수 있다. 또한 'Primary' 기능에 대한 우선순위 결정에 영향을 준다.

• 'Primary' 기능에 대한 우선순위를 결정한다. 결정된 'Primary' 기능의 우선순위와 기능들간의 관계에 따라 평가 범위를 결정한다.

• 평가 범위내의 기능들에 대하여 관련된 비기능적 요구사항들을 목록화한 후, 비기능적 요구사항을 중심으로 관련된 기능들을 열거한다. 비기능적 요구사항에 따라 열거된 기능 항목들은 품질 특성간의 관계 형성과 비기능적 요구사항에 대한 아키텍처 설계 부분을 찾을 때 활용된다.

② 제시된 아키텍처에 요구되는 품질 특성을 결정한다.

결정된 평가 범위내의 비기능적 요구사항들에 대하여 'how-property'를 결정한다. 'how-property'는 'how+부사+동사' 형태로 결정된다. 이것은 비기능적 요구사항으로부터 품질 특성을 결정하는 것을 돕고, 그러한 결정에 대한 명시적 근거를 제공한다. 각 비기능적 요구사항에 대한 'how-property'로부터 아키텍처 평가시 중요하게 다루어질 품질 특성을 결정한다. 표 6은 품질 특성과 'how-property' 간의 연결 관계를 각 품질 특성에 대한 정의[21,25]로부터 살펴본 예를 나타낸 것이다.

표 6 Qualities와 how-property 사이의 관계에 대한 예

Qualities	how-property	Qualities	how-property
Availability	how long available	Maintainability	how easily modify
Efficiency	how well utilize	Portability	how well migrate
Integrity	how well protect	Reusability	how well reuse
Inter-operability	how easily exchange	Testability	how easily test
Reliability	how reliably perform		
Robustness	how robustly function		
Usability	how easily use		

③ 결정된 품질 특성들간의 관계를 형성한다.

품질 특성간의 관계는 특정 품질 특성을 향상시킴으로써 다른 품질 특성이 어떠한 영향을 받는가로 결정될 수 있다. 따라서 상위 수준에서 결정될 수 있는 품질 특성간의 관계는 표 7에서처럼 2 가지 관계로 분류할 수 있다. 표 7에서와 같이 분류하기 위해서는 각 품질 특성에 연관된 기능 항목들간의 포함 및 중첩 관계를 조사하여 두 품질 특성의 기능 항목들간의 교집합 항목이 존재하지 않으면 'independent' 관계로 결정하고, 교집합 항목이 존재하면 'potential-dependent' 관계로 결정한다.

표 7 상위 수준에서 품질 특성간의 관계 분류

분류	내용
independent	두 품질 특성간에 서로 영향을 줄 수 있는 관계가 아님
potential -dependent	두 품질 특성간에 서로 영향을 줄 수 있는 잠재적 가능성을 가짐

3.1.3 아키텍처에 대한 템플릿 정의

평가의 대상인 소프트웨어 아키텍처 역시 다양한 형태로 입력될 수 있다. 따라서 체계적이고 일관성 있는 아키텍처 평가 수행을 위해서는 일정한 형태로 소프트웨어 아키텍처를 정의할 필요가 있다. 이를 위하여 UML을 이용하여 "4+1" 뷰 아키텍처 모델로 아키텍처를 표현한다.

① 아키텍처를 표현하는 데 있어서 필요한 뷰를 결정한다. 'Use case view'는 시스템의 주요 목적과 시나리오 집합을 나타내고, 'Logical view'는 주로 기능적인 요구사항을 반영한다. 'Development view'는 개발 환경에서의 시스템의 정적 구조를 나타내고, 'Process view'는 동시성과 동기화 측면을 잘 반영하며 'Deployment view'는 소프트웨어의 하드웨어로의 배치와 관련된 특성들을 반영해 줄 수 있다.

② UML을 이용하여 아키텍처에 대한 각 뷰를 표현한다. 각 뷰는 표 3에서 나타낸 정보를 표현한다. 그리고 이 과정에서는 제시된 아키텍처 기술 내용을 정제하거나, 필요한 아키텍처 뷰를 새롭게 정의하여 표현하는 작업이 포함될 수 있다.

3.2 평가 수행

평가 수행 단계는 아키텍처에 요구되는 품질 특성에 관련된 아키텍처 설계 결정 및 근거 데이타를 추출하고 결정하는 단계이다. 이를 위하여 주요 설계 요소를 확인하고, 그것으로부터 설계 결정 사항을 추출한다. 또한 그러한 설계 결정 사항에 대한 근거 데이타를 결정한다. 이와 같이 결정된 설계 결정 및 근거 데이타는 품질 예측과 아키텍처의 선택 및 향상에 대한 정보를 제공한다.

3.2.1 기능적 요구사항이 반영된 아키텍처 설계 부분 결정

평가 대상인 아키텍처에서 평가의 초점이 될 설계 부분들을 결정한다. 이를 위하여 다음 과정을 수행하여 기능적 요구사항이 반영된 아키텍처 설계 부분을 산출한다.

① 평가 범위 내의 'Primary' 기능에 대하여 Use case view의 Use case diagram으로부터 시스템의 주요 목적을 나타내는 Use case를 결정한다. Use case는 아키텍처 설계에 있어서 출발점이 되므로, 평가시 주요 아키텍처 설계 부분을 결정하는 데 있어서 먼저 Use case를 결정하게 된다.

② 결정된 Use case의 행동 흐름을 확인한다. 이는 각 Use case에 대하여 나타낼 수 있는 Activity diagram 또는 Sequence diagram을 이용하여 알 수 있다.

③ 'Primary' 기능과 관련된 기능 집합에 요구되는 품질 특성으로부터 필요한 뷰를 결정한다. Logical view는 기능성, Developmemt view는 유지보수성, 재사용성, 이식성, 안전성 측면을 잘 나타낼 수 있다[1,3]. 또한 Process view는 성능, 확장성, 유용성 측면을 나타낼 수 있고, Deployment view는 유용성, 신뢰성, 성능, 확장성 측면을 잘 나타낼 수 있다[1,3].

④ 각 뷰로부터 Use case에 대한 일련의 행동 흐름에 관계되는 설계 부분을 결정한다.

3.2.2 아키텍처 설계 결정에 대한 집합을 결정

아키텍처 설계 결정(Architectural Design Decisions)은 아키텍처 설계시 고려될 수 있는 설계 문제들에 대하여 관련된 설계 대안들에 대한 선택을 의미한다. 하나의 아키텍처 설계 결정은 설계 결정 변수와 설계 결정 값으로 구성된다.

아키텍처 설계 결정=설계 결정 변수-설계 결정 값

① 설계 결정 변수(Decision Variables)를 결정한다.

아키텍처 설계 단계에서 고려될 수 있는 설계 문제들에 대하여 다양한 설계 대안들이 존재할 수 있다. 설계 결정 변수는 아키텍처 설계시 고려되는 상위 수준에서의 특정 설계 문제 영역들을 말한다. 존재할 수 있는 설계 결정에 대한 문제 영역은 표 3에서 정리한 아키텍처 정보의 집합으로부터 결정된다. 상위 수준에서의 설계 결정 문제 영역에 대하여 아키텍처 설계 사항으로부터 설계 결정 변수를 추출한다.

② 설계 결정 값(Decision Values)을 결정한다.

설계 결정 값은 설계 결정 문제 영역들에 대하여 아키텍처로부터 판단될 수 있는 선택된 설계 대안들을 말한다. 따라서 각 설계 결정 변수에 대하여 선택된 설계 대안을 아키텍처 뷰로부터 결정한다.

3.2.3 아키텍처 설계 결정에 대한 근거 데이타 결정

아키텍처 설계 대안들에 대하여 내려진 설계 결정에 대하여 표 8에서 나타낸 내용으로 각 설계 결정에 대하여 합리적으로 판단해 볼 수 있는 근거 데이타를 결정하여 산출한다.

표 8 근거 데이타에 포함되는 내용

구성 요소	내용
Rationale ID	근거 데이타 식별자
Architectural design decisions	아키텍처 설계시 고려될 수 있는 설계 문제들에 대하여 관련된 설계 대안들에 대한 선택
Candidate alternative	설계 결정 부분에 대하여 가능한 대안들
Consequences	설계 결정이 가지는 특성에 미치는 영향, 관련 품질 요소에 대한 영향

또한 근거 데이타는 각각 그림 2와 같은 형식으로 작성한다.

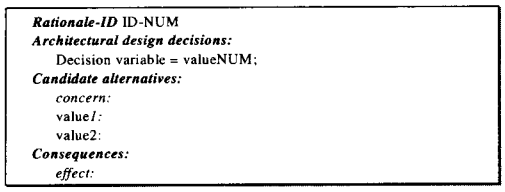

그림 2 근거 데이타를 표현하는 형식

다음은 근거 데이타의 구성 요소를 결정하는 방법을 설명한 것이다.

① Rationale-ID 결정

비기능적 요구사항과 그것에 대한 설계 결정 항목의 개수에 따라 Rationale ID가 부여된다. 즉 비기능적 요구사항 NFR1에 대하여 2 개의 설계 결정 사항이 존재한다면, 각각 RID1-1, RID1-2 형태로 부여될 수 있다. 이러한 Rationale ID는 최종 평가 결과물을 구조화할 때 이용된다.

② Architectural design decisions 결정

아키텍처 설계 결정에 대한 집합으로부터 근거 데이타를 작성하고자 하는 아키텍처 설계 결정 항목을 표시한다.

③ Candidate alternatives 결정

설계 결정 변수의 주요 초점이 무엇인지 결정하여 그림 2의 'Concern'으로 나타낸다. 다음으로 'Concern'에 따라 설계 결정 값으로 가능한 대안들을 결정한다. 이 과정에서는 후보 아키텍처가 존재한다면 그것으로부터 가능한 대안들을 쉽게 결정해줄 수 있다.

④ Consequence 결정

설계 결정과 관련된 비기능적 요구사항이 제시된 아키텍처에서 가지는 보다 구체적인 의미와 설계 대안들간의 차이를 반영하여 설계 결정에 대한 구체적 특성을 정의한다. 다음으로 결정된 특성에 대하여 기존에 존재하는 품질 평가 모델[4,26]이나 설계 이론[27,28] 등의 방법을 이용하여 설계 결정이 미치는 효과(effects)를 구한다. 더불어 잠재적 관계(potential-dependent relationship)를 가지는 품질 특성에 직접적인 관계를 가지는 설계 결정 사항들에 대한 효과를 동일한 방법으로 구한다.

3.2.4 설계 결정 변수 사이의 관계 정의

품질 특성에 영향을 주는 설계 결정 변수간의 관계를 정의하여 나타낸다. 이를 위하여 각 설계 결정에 대한 근거 데이타에 나타낸 'Consequence' 정보로부터 설계 결정 변수 사이의 관계를 결정한다. 이를 통하여 각 설계 결정 변수가 다른 설계 결정 변수의 품질 특성에 대하여 가지는 긍정적(+), 부정적(-) 관계를 나타낸다.

3.3 평가 완료

평가 완료 단계는 평가 준비 및 평가 수행 단계에서 결정된 산출물을 바탕으로 소프트웨어 아키텍처 평가시 고려했던 설계 결정들에 대하여 품질 특성에 대한 상관관계를 이용하여 그룹화한다. 또한 평가 대상이 되는 소프트웨어 아키텍처에 대하여 품질을 예측하고 아키텍처를 선택하기 위한 데이타를 생성한다.

3.3.1 설계 결정 변수들을 그룹화

평가하고자 하는 각 품질 특성에 대하여 결정된 설계 결정 변수들이 어떠한 관계를 가지는가에 따라 각 품질에 대하여 다음 두 가지 종류의 그룹으로 나누어 나타낸다.

• 'positive' 그룹

해당 품질 특성에 대하여 관계된 설계 결정이 긍정적인 효과(+)를 가지는 경우.

• 'negative' 그룹

해당 품질 특성에 대하여 관계된 설계 결정이 부정적인 효과(-)를 가지는 경우.

이와 같이 분류된 설계 결정 사항은 아키텍처를 향상시켜 나가는 데 있어서 우선순위를 부여해 줄 수 있도록 하며, 또한 특정 설계 부분의 변경이 다른 설계 부분에 대하여 어떠한 영향을 미치는가를 이해하고 아키텍처를 향상시켜 나갈 수 있도록 유도해 줄 수 있다.

3.3.2 품질 예측을 위한 데이타를 생성

평가 결과를 가시적으로 제시하고, 평가 예측에 영향을 주는 주요 결정 요소들 사이의 관계를 표현한다. 평가 예측에 영향을 주는 요소로는 품질 특성, 근거 데이타, 아키텍처 설계 결정, 설계된 요소들이 해당된다. 이러한 품질 예측 결정 요소들을 구조화된 방법으로 표현함으로써 품질 예측 뿐만 아니라 아키텍처의 향상에 중요한 자료로서 활용될 수 있다. 또한 주어진 아키텍처에 대한 점진적 평가를 가능하게 하고, 평가 범위의 확대로 인한 복잡성을 다룰 수 있다. 아키텍처에 대한 평가를

표 9 평가 결과물을 표현하는 계층 구조의 내용

계층	특징
Qualities	결정된 품질 특성과 각 품질 특성에 대해 고려된 비기능적 요구사항을 나타냄
Rationale	설계 결정에 대한 타당성을 나타냄
Architectural design decisions	설계시 고려한 설계 결정 사항(설계 결정 변수와 설계 결정 값)들을 나타냄
Sub-designs	평가시 추출되었던 설계 결정 사항에 대해서 그것이 실제 설계에 반영되어 나타난 설계 부분

통하여 최종적으로 제시되는 평가 결과물은 계층 구조로 되어 있으며 각각에 대한 설명은 표 9와 같다.

평가 결과를 구조화하는 방법은 다음과 같다.

① 4가지 종류의 계층을 정의한다. 표 9에서 나타낸 각 계층의 특징에 따라 각 계층을 구성하는 항목들을 나타낸다.

② 계층 내부의 관계(affect 관계)를 정의한다. 계층 내부에 정의할 수 있는 관계는 품질 특성간의 'affect' 관계로서, 평가 수행 단계에서 결정된 설계 결정 변수들 사이의 관계를 이용하여 결정한다.

③ 계층 간의 관계(determine, trace 관계)를 결정한다. 계층 간의 관계는 품질 특성, 근거 데이타, 아키텍쳐 설계 결정, 설계 결정체들 사이의 양방향 관계가 존재할 수 있다. 'determine' 관계는 상위 계층에서 하위 계층 방향의 결정 관계를 의미하고, 'trace' 관계는 하위 계층에서 상위 계층 방향의 추적 관계를 나타낸다.

지금까지 살펴본 바와 같이 본 논문에서 제시하는 아키텍쳐 평가 방법은 평가 목표 및 평가 범위를 결정하고, 평가 대상을 명확하게 정의한다. 이를 바탕으로 품질 예측과 아키텍쳐 선택 및 향상에 유용한 정보들을 추출하고 결정한다. 최종적으로 평가 과정에서 결정될 수 있는 아키텍쳐 정보들을 이용하여 품질 요구사항에 대한 아키텍쳐의 적합성을 판단할 수 있도록 구조화된 평가 결과물을 얻는다.

4. 사례 연구

본 연구에서 제안한 소프트웨어 아키텍쳐 평가 방법을 Box Office System[18]에 적용하였다. Box Office System은 클라이언트/서버 타입의 티켓 판매 시스템이다. 이러한 시스템에 대한 요구 사항은 그림 3과 같다.

- Customer buys tickets through the kiosk or the clerk.
- Customer buys subscriptions through the clerk.
- Customer exchanges tickets.
- Customer checks ticket's availability.
- Supervisor surveys total sales.
- Buying tickets and subscriptions make charges to the credit card service.
- Customer can sell and release tickets, make money transfers to and from the office, print their current status reports.
- Box office system must be able to handle a number of concurrent users.
- Server keeps information about each client's transactions and status.
- Each ticket has distinct row and seat number, no support for unnumbered tickets.
- Customers may have many reservations, but each reservation is made by one customer.
- Reservations are of two kinds: subscription series and individual reservations. Both reserve tickets: in one case, only one ticket; in the other case, several tickets. Every ticket is part of a subscription series or an individual reservation, but not both.
- Every performance has many tickets available, each with a unique seat number. A performance can be identified by a show, date, and time.
- Box office system must be able to handle 6 concurrent uses and about 30 transactions per second.
- We must improve reusability of components to reuse in other ticket selling systems.
- Server securely keeps information about each client's transactions and status.
- Reservation methods must be able to be changed easily.
- The methods for buying and exchanging tickets must be able to be changed easily.

그림 3 Box Office System에 대한 요구 사항

Box Office System은 티켓 구입 및 예약 서비스를 제공하고, 티켓에 대한 대금 지불은 외부 시스템의 신용카드 결제 서비스를 이용한다. 그리고 관리자가 티켓의 판매 현황을 조사할 수 있도록 티켓 판매 현황에 대한 정보를 제공한다.

4.1 평가 준비(Prepare Evaluation)

4.1.1 요구사항에 대한 템플릿 정의

Box Office System에 대한 전체 요구사항으로부터 기능적 요구사항과 비기능적 요구사항을 분리하여 나타내면 그림 4와 같다.

Functional requirements
- Customer buys tickets through the kiosk or the clerk.
- Customer buys subscriptions through the clerk.
- Customer exchanges tickets.
- Customer checks ticket's availability.
- Supervisor surveys total sales.
- Buying tickets and subscriptions make charges to the credit card service.
- Customer can sell and release tickets, make money transfers to and from the office, print their current status reports.
- Box office system must be able to handle a number of concurrent users.
- Server keeps information about each client's transactions and status.
- Each ticket has distinct row and seat number, no support for unnumbered tickets.
- Customers may have many reservations, but each reservation is made by one customer.
- Reservations are of two kinds: subscription series and individual reservations. Both reserve tickets: in one case, only one ticket; in the other case, several tickets. Every ticket is part of a subscription series or an individual reservation, but not both.
- Every performance has many tickets available, each with a unique seat number. A performance can be identified by a show, date, and time.

Non-functional requirements
- Box office system must be able to handle 6 concurrent uses and about 30 transactions per second.
- We must improve reusability of components in order to reuse in other ticket selling systems.
- Server securely keeps information about each client's transactions and status.
- Failure rate of ticketing service should not exceed 1 in 1000 requests.
- Reservation methods must be able to be changed easily.
- The methods for buying and exchanging tickets must be able to be changed easily.

그림 4 Box Office System의 기능적, 비기능적 요구사항

4.1.2 평가 협약 생성

그림 5(a)에서와 같이 분리된 기능적 요구사항으로부터 시스템의 주요 목적을 나타내는 'Primary' 기능으로 'Customer buys tickets through the kiosk or the clerk', 'Customer buys subscriptions through the clerk' 그리고 'Supervisor surveys total sales' 기능 등을 추출한다. 그리고 결정된 'Primary' 기능과 관련하여 'Mandatory' 기능과 'Optional' 기능을 분류하여 결정한다. 또한 기능들 사이에 'supported-by' 관계와 'associated-with' 관계를 나타낸다. 그리고 'Primary' 기능에 대하여 우선 순위를 부여하여 나타낸다. 'Primary' 기능의 우선 순위에 따라 결정된 평가 범위 내의 'Customer buys tickets through the kiosk or the clerk'과 관련된 기능 집합에 대하여 비기능적 요구사항을 추출하고, 비기능적 요구사항을 중심으로 기능 목록을 정리하여 나타내면 그림 5(b)와 같이 나타낼 수 있다.

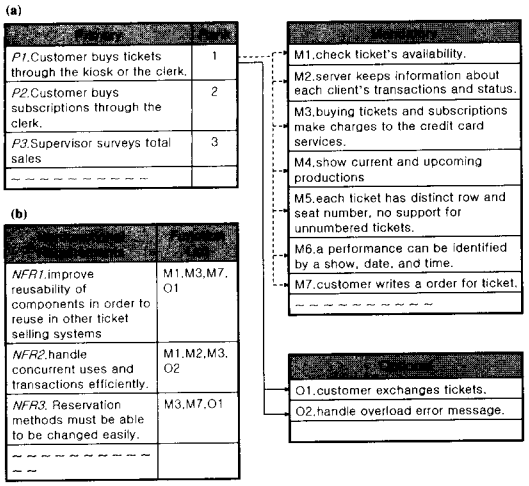

그림 5 기능 분류 및 비기능적 요구사항에 대한 기능 목록

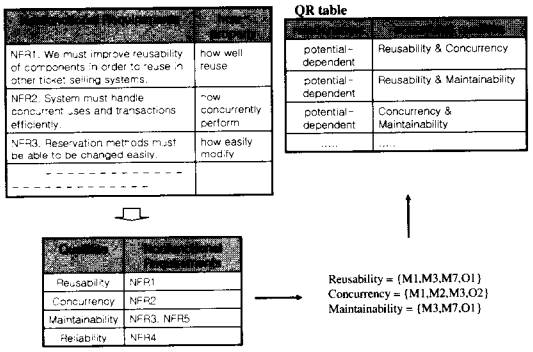

그림 6 결정된 품질 특성 및 그것들간의 관계

그림 5(b)에서 결정된 비기능적 요구사항 NFR1, NFR2, NFR3에 대하여 'how-property'를 나타내면 각각 'how well reuse', 'how concurrently perform', 그리고 'how easily modify'로 나타낼 수 있다. 이를 토대로 Reusability, Concurrency, Maintainability 등의 품질 특성을 결정하고, 각 품질 특성과 관련된 기능 목록들간의 포함관계로부터 그림 6과 같이 품질 특성간의 관계를 결정한다.

4.1.3 아키텍처에 대한 템플릿 정의

평가 대상인 소프트웨어 아키텍처는 UML을 이용하여 "4+1" 뷰 아키텍처 모델로 그림 7, 그림 8, 그림 9, 그림 10과 같이 나타낼 수 있다[18].

먼저 그림 7은 Box Office System의 Use case view를 나타낸 것이다. 그림 7의 (a)는 시스템의 주요 목적을 나타내기 위하여 Actors와 Use cases를 보여준다. 그림 7의 (b)는 Kiosk, Box Office System, Credit card service 사이에 일어나는 'Buy tickets' use case에 대한 상호작용의 흐름을 보여준다.

다음으로 그림 8은 Box Office System의 Logical view를 나타낸 것이다. 그림 8(a)의 Class diagram은 Customer, Reservation, Ticket, Performance 클래스를 포함하고 있다. 그림 8(b)의 Sequence diagram은 주요 클래스의 인스턴스와 외부 Actors 사이에서 티켓 발매를 위해 일어나는 상호작용을 나타낸다.

그림 9는 Box Office System의 Development view와 Deployment view를 나타낸 것이다. 그림 9(a) Development view는 Component diagram을 이용하여 나타내었다. Box Office System은 3 개의 사용자 인터페이스 컴포넌트(Kiosk를 이용하는 고객을 위한 것, 온라인 예약을 이용한 Clerks을 위한 것, 티켓 판매 현황

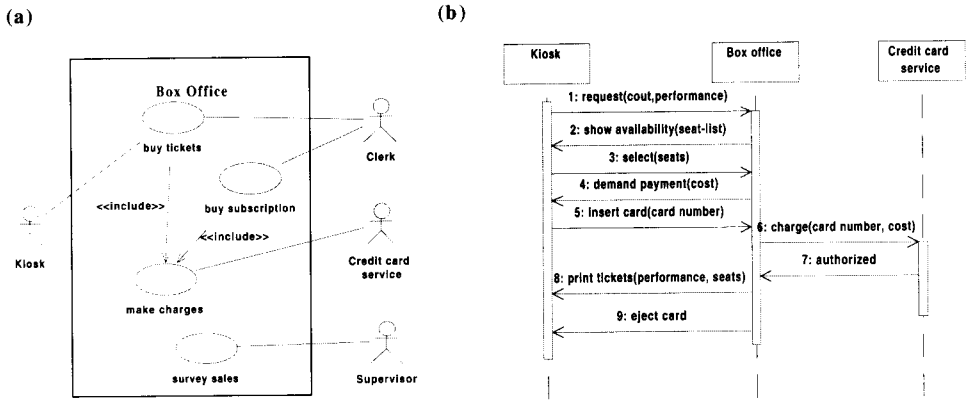

그림 7 Box Office System의 Use case view

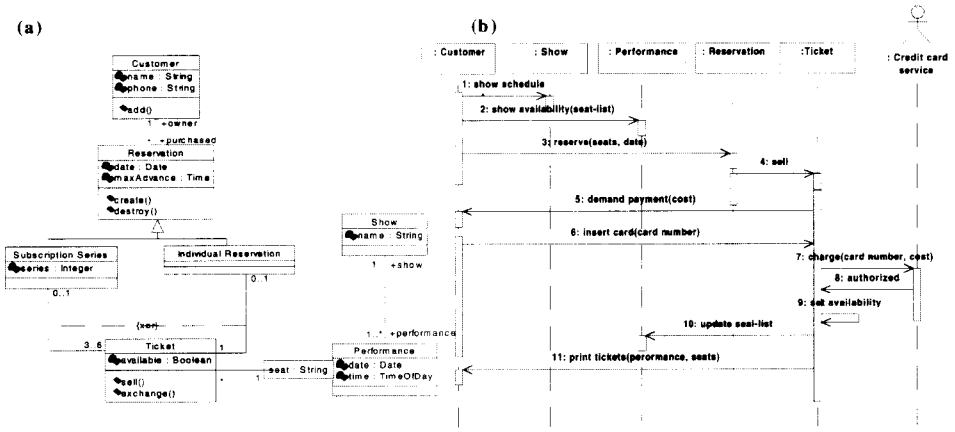

그림 8 Box Office System의 Logical view

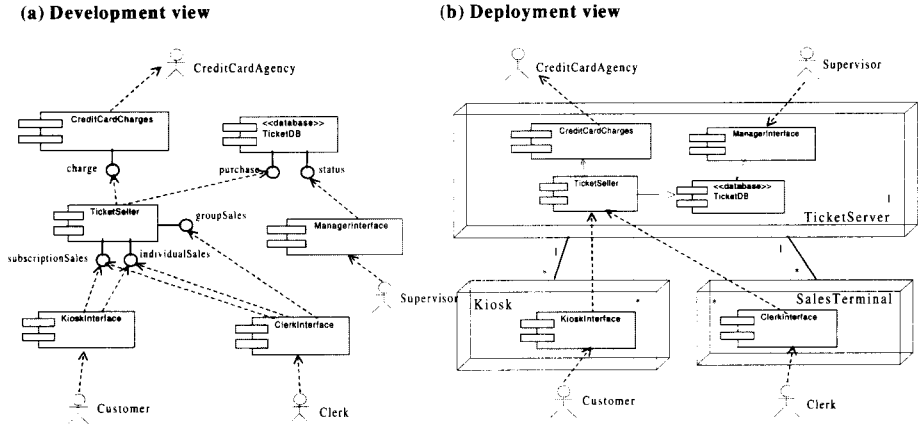

그림 9 Box Office System의 Development view와 Deployment view

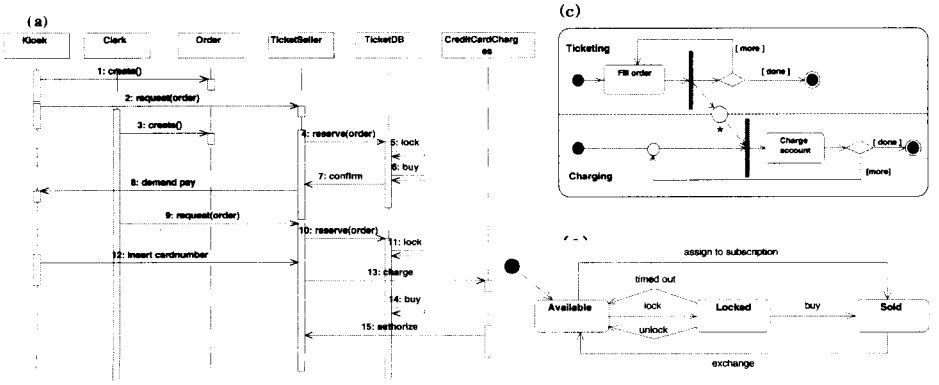

그림 10 Box Office System의 Process View

에 대하여 조회를 하는 Supervisors를 위한 것)와 티켓 발매 요구를 정렬하고 처리하는 컴포넌트, 신용 카드 결제를 처리하는 컴포넌트, 티켓 정보를 포함하는 데이타베이스 컴포넌트로 구성되어 있다. 그림 9(b) Deployment view는 Deployment diagram을 이용하여 컴포넌트의 배치를 나타낸다.

그림 10은 Box Office System의 Process view를 나타낸 것이다. 그림 10(a)는 티켓 발매 요청 및 처리가 Kiosk 또는 Clerk에 의해 동시에 일어나는 상황을 Sequence diagram을 이용하여 나타낸 것이다. 그리고 그림 10(b)는 티켓 발매 요청 및 처리 과정에서의 동시성과 동기화 측면을 Sequence diagram을 이용하여 나타내었고, 그림 10(c)에서는 티켓 발매 요청 및 처리 과정에서 동시성을 위해 티켓이 가질 수 있는 상태를 나타내었다.

4.2 평가 수행(Execute Evaluation)

4.2.1 기능적 요구사항이 반영된 아키텍처 설계 부분 결정

평가 범위 내의 'Primary' 기능인 'Customer buys tickets through the kiosk or the clerk' 기능에 대하여 아키텍처의 Use case view로부터 'buy tickets' use case를 찾아낸다. 또한 Use case view의 Sequence diagram에서 'buy tickets' use case에 대한 일련의 행동 흐름을 결정한다. 다음으로 'Customer buys tickets through the kiosk or the clerk' 기능에 대해 요구되는 재사용성, 동시성, 유지보수성과 같은 품질 특성에 따라 Logical view, Development view, Process view에서 'buy ticket' use case의 행동 흐름에 관여하는 아키텍처 설계 부분을 결정한다. 그림 11은 'Customer buys tickets through the kiosk or the clerk' 기능에 대한 아키텍처 설계 부분을 나타낸 것이다[그림 7,8,9,10 참조].

4.2.2 아키텍처 설계 결정에 대한 집합을 결정

그림 6에서 비기능적 요구사항 'NFR1'에 대해서는 Development view로부터 'DtComponentLayering'을 설계 결정 변수로 추출하였고, 'NFR2'에 대해서는 Process view로부터 'DtIndependentTasks'와 'DtSynchronization'을 설계 결정 변수로 추출하였다. 그리고 'NFR3'에 대해서는 Development view와 Logical view로부터 'DtComponentDependency'를 설계 결정 변수로 추출하였다. 이러한 설계 결정 변수에 대하여 해당 뷰로부터 설계 결정 값을 결정하였다. 그림 12는 추출된 설계 결정 변수와 설계 결정 값을 나타낸 것이다.

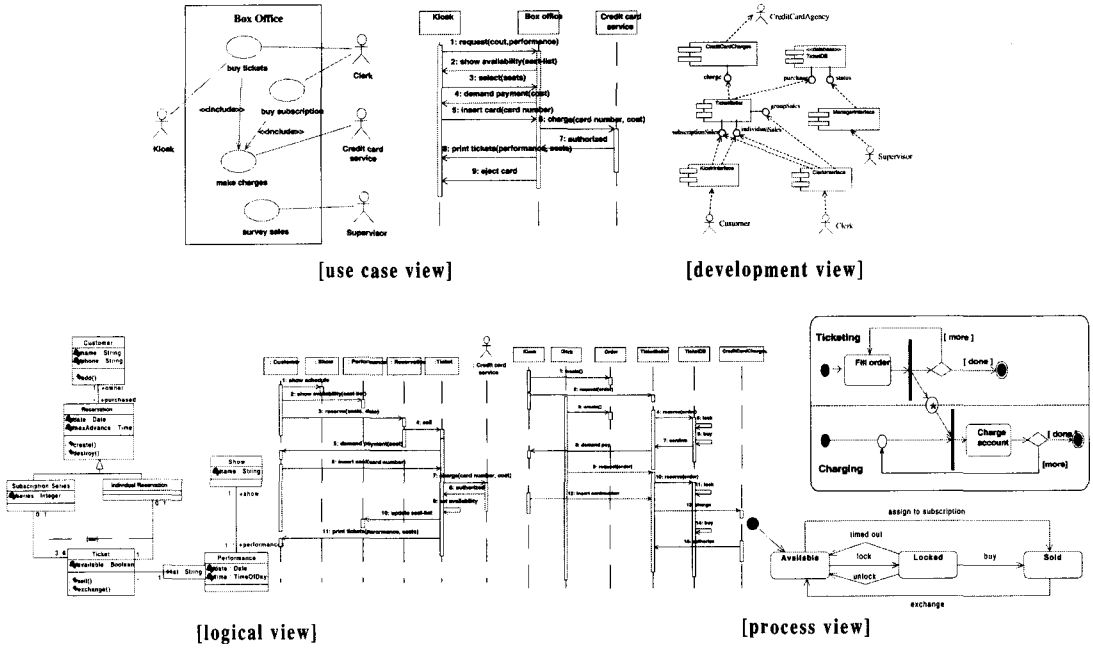

그림 12 'buy tickets' use case에 대한 아키텍처 설계 부분

Decision variables

NFR1	DtComponentLayering	계층 Component 역할에 대한 결정
NFR2	DtIndependentTasks	Parallel하게 수행될 수 있는 tasks 결정
NFR2	DtSynchronization	Tasks 사이의 synchronization 결정
NFR3	DtComponentDependency	각 Component interfaces 결정 및 그것을 통한 Components사이의 연결관계 결정

Decision values

DtComponentLayering	3-layering: Interfacing, Processing, Storing components
DtIndependentTasks	2-Tasks: Charging, Ticketing tasks
DtSynchronization	Existence of unlimited synch state
DtComponentDependency	3-Reservation interfaces: subscriptionSales, groupSales, individualSales,

그림 13 설계 결정 변수와 설계 결정 값

4.2.3 아키텍처 설계 결정에 대한 근거 데이타 결정

그림 13은 각 아키텍처 설계 결정에 대한 근거 데이타를 나타낸 것이다. 그림 13(a)의 'Rationale RID1'은 설계 결정 변수 'DtComponentLayering'에 대한 것으로, 이것은 재사용성과 직접적으로 관련된다. 이 변수의 주요 'Concern'은 기능성의 분리인 것으로 결정되었다. 이러한 'Concern'에 대하여 가능한 대안들로는 사용자와의 상호 작용을 다루는 컴포넌트와 시스템 내부 컴포넌트로 양분하는 방법과 사용자와의 상호작용 및 처리 컴포넌트를 결합하고 저장 컴포넌트를 따로 분리하는 방법이 있을 수 있다. 그리고 사용자와의 상호작용 컴포넌트와 처리 컴포넌트, 저장 컴포넌트로 각각 기능을 분리하는 방법이 있다. 그 중에서 선택된 방법은 세 번째 방법이며, 이 방법은 각 계층의 컴포넌트의 재사용성을 증가시키는 효과를 가진다[28].

그림 13(b)의 'Rationale RID2-1'은 설계 결정 변수 'DtIndependentTasks'에 대한 것으로, 이것은 동시성과 직접적으로 관련된다. 이 변수의 주요 'Concern'은 단일 프로세싱 노드에서 개별적으로 스케쥴링 될 수 있는 작업 단위로의 분리인 것으로 결정되었다. 이러한 'Concern'에 대하여 가능한 대안들은 티켓 주문 및 발권과 대금 청구 작업을 분리한 방법과 두 가지를 하나의 작업 단위로 수행하는 방법이 있을 수 있다. 이들 대안 중 선택된 방법은 첫 번째 방법이다. 이 방법은 개별적으로 수행 가능한 두 작업을 분리하여 작업 단위를 작게 하고 개별 쓰레드에서 수행될 수 있도록 함으로써, 다수의 사용자에 의한 동시성을 향상시킨다[18].

그림 13(c)의 'Rationale RID2-2'는 설계 결정 변수 'DtSynchronization'에 대한 것으로, 역시 동시성과 직접적으로 관련되며, 독립적으로 수행되는 각 실행 영역을 구성하는 상태에 대한 결정이 주요 'Concern'이다. 이에 대하여 가능한 대안들은 복합된 상태와 unlimited Synch 상태를 두는 방법과, 세분화된 상태와 bounded Synch 상태를 두는 방법이 있을 수 있다. 이들 대안 중 선택된 대안은 첫 번째 방법이며, 이 방법은 동시성을 보다 안전하게 지원할 뿐만 아니라 작업 전환에 따른 성능 저하를 최소로 한다. 그러나 교착 상태의 가능성이 존재한다[18].

그림 13(d)의 'Rationale RID3'은 설계 결정 변수 'DtComponentDependency'에 대한 것으로, 이것은 유지보수성과 직접적으로 관련된다. 이 변수의 주요 'Concern'은 예약 방법 변경의 용이성인 것으로 결정되었다. 이에 대하여 가능한 대안으로는 예약 방법에 따라 서로 다른 인터페이스를 정의하는 방법과 두 개 또는 하나의 인터페이스로 통합시킨 방법이 있을 수 있다. 이들 대안 중 선택된 방법은 현재의 예약 방법에 따라 인터페이스를 분리한 방법이다. 컴포넌트의 인터페이스가 가지는 역할을 작게 하고, 각 인터페이스의 기능을 잘 분리하는 것은 특정 인터페이스의 변경이 다른 인터페이스를 사용하는 컴포넌트에 영향을 미치지 않게 한다. 그러므로 일부 기능의 변경이 용이하며, 또한 그러한 컴포넌트의 재사용성을 증가시킨다[18].

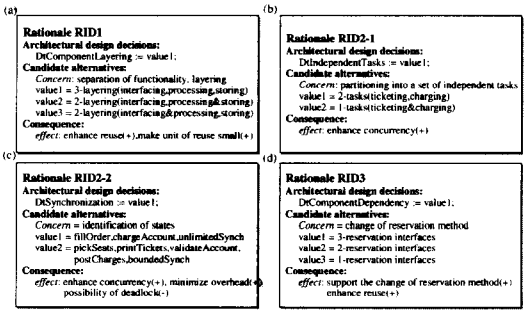

그림 14 아키텍처 설계 결정에 대한 근거 데이타

4.2.4 설계 결정 변수 사이의 관계 추출

설계 결정 변수 사이의 관계에 있어서는 그림 13에서 결정된 각 설계 결정에 대한 근거 데이타에서의 Consequence를 바탕으로 'DtComponentDependency'가 'DtComponentLayering'에 대한 설계 결정에 양성적(+) 영향을 가진다는 것을 알 수 있다. 그리고 'DtIndependentTasks'와 'DtSynchronization'에 대한 설계 결정이 상호 양성적(+) 영향을 준다. 따라서 표 10과 같은 결과를 얻을 수 있다.

표 10 추출된 설계 결정 변수 사이의 관계

Decision variables	Related decision variables	Qualities
DtComponentLayering	-	-
DtIndependent Tasks	DtSynchronization	Concurrency(+)
DtSynchronization	DtIndependent-Tasks	Concurrency(+)
DtComponent Dependency	DtComponent-Layering	Reusability(+)

4.3 평가 완료(Complete Evaluation)

4.3.1 설계 결정 변수들을 그룹화

표 11은 각 품질 특성에 대하여 'positive', 'negative' 영향을 가지는 설계 결정들을 그룹화하여 나타낸 것이다.

표 11 설계 결정 변수의 그룹화

Qualities	Groups	Variables
Reusability	positive	DtComponentLayering, DtComponentDependency
	negative	-
Concurrency	positive	DtIndependentTasks, DtSynchronization
	negative	-
Maintainability	positive	DtComponentLayering
	negative	-

4.3.2 품질 예측을 위한 데이타 생성

그림 14는 최종적으로 생성되는 품질 예측 데이타를 나타낸 것이다. 이것은 아키텍처 평가 과정에서 결정된 품질(*Qualities*), 근거 데이타(*Rationale*), 아키텍처 설계 결정(Architectural design decisions), 그리고 초점이 된 아키텍처 부분 설계 사항(Sub-designs)들을 계층 구조로 나타낸다. 품질 계층에는 주어진 아키텍처에 대해 요구되는 품질 특성인 재사용성, 동시성, 유지보수성 등의 항목이 나타나고, 유지보수성에 대한 설계가 재사용성에 영향을 줄 수 있음을 'affect' 관계로 나타내었다. 아키텍처 설계 결정 계층은 'DtComponentLayering', 'DtIndependentTasks', 'DtSynchronization', 그리고 'DtComponentDependency' 등 설계 결정 변수만을 간단하게 나타내었다. 근거 데이타 계층에는 설계 결정들에 대한 4 개의 근거 데이타를 식별자로 나타내었고, 아키텍처 설계 결정을 추출한 아키텍처 설계 사항들이 아키텍처 부분 설계 계층에 나타난다. 각 계층은 관련 정보들 사이의 결정 및 추적 관계로 연결된다. 이를 통하여 3 가지 품질 특성에 대한 주어진 아키텍처의 적합성을 판단할 수 있다.

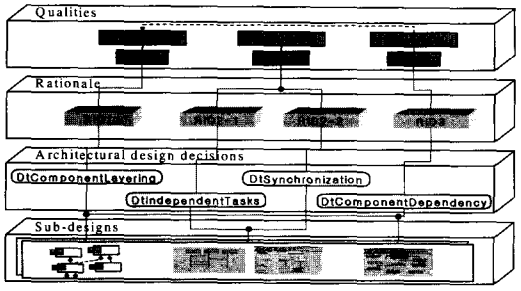

그림 15 품질 예측을 위한 평가 결과물

5. 결론 및 향후 연구 과제

소프트웨어 아키텍처 평가를 위해서는 아키텍처에 대하여 기대되는 품질 요구사항이 무엇인지를 결정하고, 평가 대상인 아키텍처를 분명하게 정의하는 것이 필요하다. 또한 정의된 입력물로부터 최종 평가 결과물에 이르기까지의 체계적이고 객관적인 프로세스가 필요하며, 최종 평가 결과물에 대한 분명한 정의가 요구된다. 그러나 기존의 아키텍처 평가 방법에서는 평가의 초기 입력물이 체계적으로 고려되지 않음으로써 평가 목표 및 평가 범위, 평가 대상이 불분명하다. 또한 절차적인 측면에서는 주관적인 요소에 의해 많이 좌우되거나 혹은 평가 프로세스를 별도로 정의하지 않고 있다. 그리고 최종 평가 결과물에 대한 명시적 표현이 부족하여 품질 예측과 아키텍처 선택 및 향상에 있어서 도움을 기대하기 어렵다. 본 논문에서 제안한 아키텍처 평가 방법은 요구사항 및 아키텍처를 체계적으로 다루고 정의하였다. 그리고 평가 절차에 있어서는 정의된 입력물을 바탕으로 아키텍처의 품질 예측과 관련된 요소들인 품질, 근거 데이타, 설계 결정, 부분 설계 등을 결정하기 위한 프로세스를 정의하였다. 또한 평가 결과물에 대한 표현 방법을 명확하게 제시하였다. 마지막으로 제안한 방법을 Box Office System의 아키텍처 평가에 적용해 봄으로써, 제안한 방법이 아키텍처 평가에 효과적으로 적용 가능함을 확인하였다.

향후 연구 과제로는 다양한 아키텍처에 대한 사례 연구를 통하여 제안한 방법을 분석하고 방법의 절차 및 산출물에 대한 보완이 필요하다. 또한 기존 품질 평가 모델들과의 효과적인 연계 방안에 대한 고려가 필요하고, 평가 프로세스를 지원하기 위한 도구 개발에 대한 노력이 요구된다.

참 고 문 헌

[1] Clements, P. and Northrop, L., "Software Architecture: An Executive Overview," CMU/SEI-96-TR-003, Carnegie Mellon University, February 1996.

[2] Bass, L., Clements, P., and Kazman, R., Software Architecture in Practice, Addison-Wesley, 1998.

[3] Kruchten, P., "The 4+1 View Model of Software Architecture," IEEE Software, Vol. 12, No. 6, pp.42-50, November 1995.

[4] Kazman, R. et al., "The Architecture Tradeoff Analysis Method," The 4th IEEE International Conference on Engineering of Complex Computer Systems, pp.68-78, 1998.

[5] Kazman, R., Abowd, G. Bass, L., and Clements, P., "Scenario-Based Analysis of Software Architecture," IEEE Software, pp.47-55, November 1996.

[6] Kazman, R., Klein, M., and Clements, P., "Evaluating Software Architectures for Real-Time Systems," Annals of Software Engineering, pp.1-27, 1999.

[7] Allen, R., "A Formal Approach to Software Architecture," CMU-CS-97-144, Carnegie Mellon University, May 1997.

[8] Medvidovic, N., "A Classification and Comparison Framework for Software Architecture Description Languages," UCI-ICS-97-02, University of California, Irvine, February 1996.

[9] Abowd, G. et al., "Recommended Best Industrial Practice for Software Architecture Evaluation," CMU/SEI-96-TR-025, Carnegie Mellon University, January 1997.

[10] Bosch, J. and Molin, P., "Software Architecture Design: Evaluation and Transformation," Proceedings of IEEE Conference and Workshop on Engineering of Computer-Based Systems, pp.4-10, 1999.

[11] Barbacci, M.R., Klein, M.H., and Weinstock, C.B., "Principles for Evaluating the Quality Attributes of a Software Architecture," CMU/SEI-96-TR-036, Carnegie Mellon University, May 1997.

[12] Clements, P., Bass, L., Kazman, R., and Abowd, G., "Predicting Software Quality by Architecture-Level Evaluation," The 5th International Conference on Software Quality, pp.485-498, 1995.

[13] Bredemeyer Consulting, http://www.bredemeyer.com/definiti.htm, 2001.

[14] CMU/SEI http://www.sei.cmu.edu/architecture/definitions. html, 2001.

[15] Perry, D.E. and Wolf, A.L., "Foundations for The Study of Software Architecture," ACM SIGSOFT Software Engineering Notes, Vol. 17, No. 4, pp.40-52, October 1992.

[16] Garlan, D. and Shaw, M., "An Introduction to Software Architecture," Advances in Software Engineering and Knowledge Engineering, Series on Software Engineering and Knowledge Engineering, Vol. 2, pp.1-39, World Scientific Publishing Co., 1993.

[17] Eriksson, H. and Penker, M., UML Toolkit, Addison-Wesley, 1998.

[18] Rumbaugh, J., Jacobson, I, and Booch, G., The Unified Modeling Language Reference Manual, Addison-Wesley, 1999.

[19] Hofmeister, C., Nord, R., and Soni, D., Applied Software Architecture, Addison-Wesley, 2000.

[20] Kotonya, G. and Sommerville, I., "Requirements Engineering with Viewpoints," Software Engineering Journal, pp.5-18, January 1996.

[21] Wiegers, K.E., Software Requirements, Microsoft Press, 2000.

[22] Jones, L.G. and Kazman, R., "Software Architecture Evaluation in the DoD Systems Acquisition Context," http://interactive.sei.cmu.edu, SEI Interactive, December 1999.

[23] Bosch, J., Design and Use of Software Architectures, Addison Wesley, 2000.

[24] Clements, P., "A Survey of Architecture Description Languages," Proceedings of the 8th International Workshop on Software Specification and Design, pp.1-26, March 1996.

[25] IEEE Standard Glossary of Software Engineering Terminology, IEEE Std. 610.12-1990.

[26] Iannino, A., "Software Reliability Theory," Encyclopedia of Software Engineering, John Wiley & Sons, pp.1237-1253, February 1994.

[27] Schmidt, D., Stal, M., Rohnert, H., and Buschmann, F., Pattern-Oriented Software Architecture-Patterns for Concurrent and Networked Objects, John Wiley & Sons, 2000.

[28] Buschmann, F., et al., Pattern-Oriented Software Architecture-A System of Patterns, John Wiley & Sons, 2000.

최 희 석

1998년 2월 부산대학교 컴퓨터공학과(학사). 2000년 2월 부산대학교 컴퓨터공학과(석사). 2000년 3월 ~ 현재 부산대학교 컴퓨터공학과 박사과정. 관심분야는 소프트웨어 아키텍처, 컴포넌트 기반 소프트웨어 개발 방법론, 분산 컴포넌트, 컴포넌트 저장소, 도메인 공학, 소프트웨어 재사용 등

염 근 혁

1985년 서울대학교 계산통계학과(학사). 1992년 Univ. of Florida 전산학과(석사). 1995년 Univ. of Florida 전산학과(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 조교수. 부산대학교 컴퓨터 및 정보통신 연구소 연구원. 관심분야는 컴포넌트 기반 소프트웨어 개발, 도메인 공학, 소프트웨어 아키텍쳐, 객체지향 소프트웨어 개발방법론, 요구 검증, 분산 컴포넌트, 소프트웨어 재사용 등