C&C 아키텍처 기반의 객체지향 설계

(Developing Object Oriented Designs from Component-and-Connector Architectures)

박 형 일 † 강 성 원 †† 최 윤 석 ††† 이 단 형 ††
(Hyoungiel Park) (Sungwon Kang) (Yoonseok Choi) (Dan H Lee)

요 약 본 논문에서는 컴포넌트와 커넥터 아키텍처로부터 세부적인 객체지향 설계를 개발하는 체계적인 방법을 제안한다. 제안된 방법은 아키텍처 모델에서 세부설계모델을 도출하는 과정에 중간모델을 도입하여, 두 모델간에 놓여 있는 추상화 수준의 격차를 줄임으로써 세부 설계도출을 용이하게 한다. 본 논문에서는 제안된 방법의 효과를 검증하기 위하여 제안방법을 산업계의 소프트웨어개발과제에 적용하고 컴포넌트와 커넥터 아키텍처가 지원하는 품질속성들이 궁극적으로 세부 설계에서도 보전되고 있음을 확인한다.
키워드 : 컴포넌트와 커넥터 아키텍처, 중간모델, 객체지향 설계

Abstract In this paper, a systematic approach of developing detail OO designs from Component-and-Connector Architectures (CCAs) is proposed. In this approach, an intermediate model between the architecture model and the detail design model specified with class diagrams or sequence diagrams is introduced to narrow the wide gap between the two abstraction levels. Once a CCA is designed, candidate classes and their relationships are identified per each architectural element. In order to show the efficacy of this approach, we apply it to an industry software development project and verify that quality attributes supported by the CCA are equally maintained by the detail design.
Key words : Component & Connector Architecture, Intermediate Model, Object-Oriented Design

1. 서 론

많은 개발자들이 소프트웨어 아키텍처를 나타내기 위하여 아키텍처기술언어(ADL: Architecture Description Language) 대신에 객체지향설계[1)] 표기법을 쓰고 있다. 비록 일부 객체지향 설계개념들은 아키텍처 설계의 쟁점들을 다루고 있고 또한 개발자들이 그것을 사용하고 있으나, 이 둘 개념은 상당한 차이를 가지고 있다.

이 논문에서는 컴포넌트와 커넥터(C&C: Component & Connector) 아키텍처[1]로 표현된 아키텍처 설계로부터 객체지향 설계를 체계적으로 개발하는 방법을 제안한다. 이 방법에서는 아키텍처 모델과 객체지향설계 사이에 두 추상화 수준의 차이를 줄이기 위하여 중간단계 모델을 도입한다. 본 논문에서는 제안된 방법의 효과를 검증하기 위하여 제안방법을 산업계의 소프트웨어개발 과제에 적용하고 C&C 아키텍처가 지원하는 품질속성들이 궁극적으로 세부 설계에서도 보전되고 있음을 확인한다.

본 논문은 다음과 같이 구성되어 있다. 본 논문의 제2절에서는 하나의 예제로 소프트웨어개발 과제인 Meta-directory 시스템 소개하고 Metadirectory시스템 개발 중 아키텍처 설계과정에 부딪힌 문제점들을 지적한다. 제3절에서는 그러한 문제들을 해결하기 위한 어떤 대안들이 있고 그들이 왜 부적절한 대안인지를 설명한다. 제4절에서는 그러한 문제들에 대한 해결책으로 중간모델 도입 기법을 소개한다. 제5절에서는 제안방법을 Meta-directory 과제에 적용하고 그 효과를 보인다. 마지막으로 제6절에서는 본 논문의 결론을 맺는다.

1) 이 논문에서는 객체지향 설계와 객체지향 상세설계를 서로 구별 없이 사용 한다.

† 비 회 원 : 티맥스소프트 컨설팅실 선임
hipark@icu.ac.kr
†† 종신회원 : 한국정보통신대학교 공학부 교수
kangsw@icu.ac.kr
danlee@icu.ac.kr
††† 비 회 원 : 한국정보통신대학교 공학부 연구원
yschoi@icu.ac.kr
논문접수 : 2005년 8월 26일
심사완료 : 2007년 2월 6일

2. Metadirectory 과제

Metadirectory 과제는 분산된 개별 시스템에 중복을 허용하여 저장된 데이타를 일관성 있게 유지하는 시스

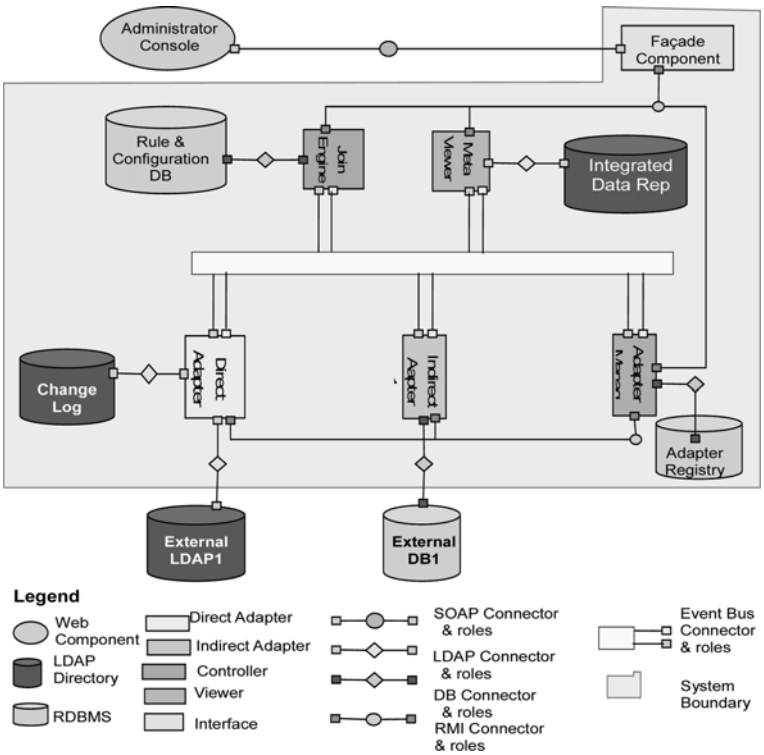

그림 1 Metadirectory 시스템의 C&C 아키텍처

템을 구축하는 과제이다. 이러한 시스템을 통해 데이타에 접근하고자 하는 사용자들은 다양한 저장소들에 분산되어 있는 데이타를 통일된 관점으로 볼 수 있다. 이 프로세스를 위하여 5명의 소프트웨어 엔지니어 팀이 각 1년 동안 개발을 수행하였다. 각 구성원들이 설계와 구현을 비롯한 모든 단계의 개발작업에 참여 하였다. 기본적인 개발 프로세스로 Unified Process(UP)를 채택하였고, 팀의 크기와 기간을 고려하여, UP가 제시하는 아키텍처 도출보다 더 나은 아키텍처 설계방법을 채택하기 위하여 프로세스를 수정하였다.

아키텍처 설계를 위하여 시각적 표현과 제약사항 및 데이타 흐름의 표현을 지원하는 Acme[2] [1]를 아키텍처 설계 툴로 선택하였다. Acme는 그림 1과 같이 컴포넌트와 커넥터의 관점에서 구조를 표현하도록 하여 준다.

이 구조는 소프트웨어 산출물이 어떤 모습을 가졌으며 어떻게 작동 할 것인지를 전체적인 관점에서 보여주기 때문에 도입(Inception)과 정제(Elaboration) 단계에서 고객과 대화하는 도구, 혹은 개발자간의 의사소통의 중요한 도구로 사용하였다. 따라서 각 개발자들은 C&C 아키텍처가 내포하는 품질속성과 제약사항들이 뒤이은 개발단계들에서 변경이나 왜곡 없이 유지되기를 바랬다. 그러나 개발자들은 C&C아키텍처로부터 객체지향의 세부설계를 만들어 가는 과정에 여러 가지 어려운 문제에 부딪혔다.

C&C아키텍처로부터 객체지향의 세부설계를 만들어 가는 과정에 부딪히게 된 여러 가지 문제점들은 다음과 같다:

(P1) C&C 아키텍처는 UP와 조화가 되지 않는다. UP의 산출물들은 객체지향의 관점에서 잘 정의가 되어 있다. 그러나 아키텍처 관련 산출물은 이러한 객체지향 기반의 산출물들과의 관계가 불분명하였기 때문에, Acme 관련 아키텍처 산출물을 프로세스에 연결하기 위하여 UP를 변경하였지만 효과적이지 못했다. 아키텍처의 특정 부분으로부터 어떻게 객체지향 상세 설계를 이끌어 내는지가 개발자의 주관적 판단에 따라서 결정되어서 일관성 있는 개발이 힘들었다.

(P2) C&C 아키텍처의 구현 방법에 대하여 일관성 유지가 힘들었다.

C&C 아키텍처의 요소들을 어떻게 구현할 수 있는지가 명시적으로 정의가 되지 않았기 때문에 각 팀 구성원들은 요소들을 어떻게 구현할 것 인가에 대하여 의견

2) Acme는 서로 다른 ADL로 작성된 명세들의 맵핑을 지원하도록 설계된 아키텍처 교환 언어이다.

의 일치가 어려웠다. 예를 들면 메시지의 입력을 기다리는 포트를 한 개발자는 독립적인 쓰레드를 써서 구현하였고 같은 방식으로 구현이 예상되는 다른 포트를 다른 개발자는 Observer 패턴[3]을 써서 구현이 하였다.

(P3) 아키텍처 스타일들을 구현하기 위한 명확한 프로세스가 존재하지 않는다. 객체지향 설계에서 클래스 다이어그램으로 표현된 것과 C&C 아키텍처에서 표현된 것은 서로 다른 개념임을 고려할 필요가 있다. 소프트웨어 아키텍처에 대한 개념이 성숙해지면서 아키텍처를 구축하는 방법이 특정 패턴들(예를 들어 “Pipe & Filter”, “Blackboard”, “Publish & Subscribe” 등)을 사용하는 것이 일반화되었다[2]. C&C 아키텍처는 설계 프로세스와 관련이 됨으로 이러한 아키텍처 스타일들이 추후 객체지향 설계와 같은 더 세부적인 설계로 구체화되어야 한다. 그러나 아키텍처 스타일들로부터 객체지향 설계로의 변환 방법이 명확히 정의되어 있지 않았다. 어떤 개발자는 어떤 설계 패턴을 이용하여 아키텍처 스타일을 구체화를 하는 반면에 다른 개발자들은 검증된 임의의 방법을 이용하여 같은 아키텍처 스타일을 구체화하고자 하였다.

(P4) 요구사항이 구현되었는지에 대한 검증의 쉽지 않았다. C&C 아키텍처는 기능적 요구사항뿐 아니라 비기능적 요구사항도 반영을 한다. 요구사항을 기반으로 컴포넌트, 커넥터, 제약사항 그리고 스타일 등을 가진다. 고객과 합의하여 확정된 C&C 아키텍처를 이용하여 개발을 하고 테스트까지 거쳤지만, 아키텍처에 내포된 세부사항들이 객체지향 설계에 어떻게 반영되었는지 일관성 있게 확인하기 어려웠다. 즉 객체지향 설계의 어떤 클래스가 어떤 품질이나 기능 요소를 위해 사용되었는지 명시적으로 나타나지 않았다.

3. 관련 연구

아키텍처 설계의 세계와 객체지향 설계의 세계를 연결하기 위한 많은 연구들이 있었다. 그런 연구들 중 대부분은 연구들은 객체지향 설계 툴을 사용하여 아키텍처를 나타내던가 또는 반대로 아키텍처기술언어를 이용하여 객체지향 설계를 하려고 한 경우였다. 이 절에서는 어떠한 연구들이 진행되어 왔으며 이러한 연구들이 어떠한 한계성이 있는지에 대하여 논의한다.

ADL을 UML로 맵핑

Wright[3], Acme 등과 같은 다양한 ADL들이 제안되었다. 다양한 종류의 ADL들이 관심을 갖는 종류의 정보는 적용 도메인, 시스템 통합의 스타일 또는 특정 속성들의 집합들이다. 이러한 공통적인 특징들을 구현하기 위해서 ADL을 세부적인 UML 설계에 맵핑을 하는 방법이 시도 되었다[5]. UML 2.0에서는 기존 버전에 문제점으로 제시되던 커넥터, Port등의 아키텍처 요소에 대한 표기법이 추가되었으나[6], 이러한 아키텍처 표기법에서 세부적인 설계에 필요한 구현 가능한 레벨의 클래스 다이어그램이나 순차 다이어그램을 어떻게 더 도출 할 수 있는가 하는 방법론적인 측면은 여전히 문제로 남아있다. 또한 확장된 프로화일을 사용하여 직접 ADL로부터 UML로 변환하는 접근법 역시 만족스럽지 못했다[7].

C&C 아키텍처의 경우는 기능/비기능적 요구사항 그리고 제약사항을 반영하기 위해서 소프트웨어의 구조를 컴포넌트와 커넥터로 표현하는 반면, 객체지향 설계는 주로 기능적 요소를 객체들의 다양한 상호작용으로 나타낸다. 전자에서 프로토콜 또는 데이타와 컨트롤의 복잡한 상호작용을 하나의 커넥터로 나타낼 수 있는 반면에 후자에서는 객체들간의 복잡한 상호작용을 통하여 프로토콜을 나타내야 한다. 이러한 두 방식간의 근본적인 차이 때문에 직접적인 맵핑 방법들은 문제가 있다.

컴포넌트 기반구조(Component Based Infrastructure)

COM, CORBA[8], Enterprise Java Beans(EJB), 그리고 Web Services[9]등과 같은 컴포넌트 기반의 인프라구조들은 naming, transaction 그리고 분산환경의 컴포넌트 기반 프로그램등과 같은 다양한 서비스를 제공한다. 이러한 인프라구조들은 개발자들에게 표준적이고 일관성 있는 상위 수준의 인터페이스를 제공함으로써 시스템 개발이 쉽게 되고 재사용이 되며 다른 환경으로 이식이 가능하도록 한다. 또한 자주 사용되는 서비스들에 대한 인터페이스를 제공하여 반복되는 코드를 줄이고 빠른 시스템 개발을 도울 수 있다. 많은 소프트웨어 엔지니어들은 전체적인 아키텍처에서 이러한 인터페이스들을 활용하여 복잡한 프로토콜들을 숨기고 빠르고 안정적으로 개발이 되도록 노력해왔다.

그러나 CORBA 또는 COM과 같은 컴포넌트 기반구조들은 명시적으로 소프트웨어 아키텍처를 기술하는 방법을 제공하지 않는다. 즉 아키텍처 설계를 구현 수준의 기반구조의 설계로 시야를 좁힘으로써 여러 참여자의 요구사항을 창의적으로 반영하는 아키텍처 설계과정에는 적절할 가이드를 주지 못하고 있다[10]. Web Services는 전체적인 아키텍처의 설계를 가능하게 함으로써 아키텍처 수준의 추상화를 제공하지만 설계 단계에서 SOAP, WSDL 또는 BPEL과 같은 표준들과 함께 Web Services 플랫폼에 의존한다. 이것은 도메인에 속하는 모든 컴포넌트들이 SOAP를 사용하고 그것의 인터페이스를 WSDL를 사용하여 공개하는 Web Services로 정의되어야 한다는 것을 의미하고 이는 일반적인 설

3) 디자인 패턴의 한 종류이다.

계 개념이 되기에는 매우 제약적이다.

ArchJava

소프트웨어 아키텍처인 구조들을 Java 소스코드로 나타내기 위해서 Java를 확장한 ArchJava가 개발되었다. ArchJava는 컴포넌트, 커넥터 그리고 포트들을 지원하기 위해서 새로운 언어 구조물들을 추가하여 개발자들이 소프트웨어 구조를 식별할 수 있도록 하였다 [11]. ArchJava의 코드는 각 아키텍처 요소들을 위한 컴포넌트 클래스들이 정의되어 있으며 이들을 활용하여 프로그램 구현을 시작하기 전에 ArchJava로 아키텍처를 명세하고 타입 체크를 할 수 있도록 하고 있다.

그러나 주요한 시스템 행위나 아키텍처 스타일등의 주요한 아키텍처 속성들에 대해 구현요소들의 맵핑을 지원하지 못한다. 실용적인 관점에서 우리가 아키텍처 수준의 컴포넌트에 대한 세부 설계할 경우, 중요한 고려사항은 아키텍처가 지원하는 품질속성들이 클래스들과 그들의 관계들을 통해 세부 설계에서도 지원되어야 한다는 것이다. ArchJava에서는 클래스 수준의 설계를 이끌 수 있는 어떠한 방법도 제시하지 않는다. 만약 개발자가 아키텍처 요소를 기존의 클래스와 클래스들간에 상속관계를 사용하여 세부 설계를 하고 싶다면 아키텍처 요소로부터 세부 설계의 과정이 미리 정해져 있는 ArchJava를 이용하기 힘들 것이다. 비록 ArchJava는 Java 언어를 아키텍처 개념으로 확장하였지만 아키텍처 개념을 객체지향 세계로 맵핑하는 방법은 제시하고 있지 못하다.

4. 중간 모델 도입 기법

중간모델 도입 기법을 설명하기에 앞서 객체지향 설계와 아키텍처 스타일간에는 직접적인 관계가 존재하지 않음을 주목하여야 한다[12]. 이는 방법론적인 측면과 표기법에 대한 의미적 차이에 기인한다. 먼저 방법론적인 측면에서 Unified Process등의 대표적인 객체지향 설계 방법론에서는 컴포넌트의 설계가 세부 디자인 단계에서 클래스들의 식별 후에 응집도와 종속도를 고려하여 설계가 되며 커넥터의 설계 역시 주로 객체들의 복잡한 상호작용을 숨기기 위한 클래스의 설계로 세부 디자인 단계에서 고려가 된다. 반면에 C&C 아키텍처의 설계는 개발 초기단계에서 고객과의 효율적인 의사통신 수단으로 활용이 되며 고객이 원하는 품질을 반영하기 위한 아키텍처적인 주요한 사항들을 결정하는데 활용이 된다. 또한 표기법에 대한 의미적 차이의 경우 여러 문제점을 가지고 있으나 대표적인 사례로써 UML 2.0에서 제시하는 커넥터는 자신의 행위 및 서브타입을 가지지 못하나 C&C 아키텍처에서는 커넥터는 제일의 모델요소로써 자신의 행위 및 서브타입을 가진다[13].

이러한 차이는 표 1에 제시된 산출물들로부터 알 수 있다. 아키텍처 설계와 클래스 설계에서 전자는 전체적인 구조와 시스템의 속성들을 고려하는 반면에 후자는 객체지향 언어로 구현 가능한 세부 설계에 중점을 둠으로써 근본적인 차이를 보인다.

쓰임새(Use case)들로부터 클래스다이어그램을 도출하는 객체지향 설계과정에는, 쓰임새를 일반화하여 객체를 도출하거나 그들간에 개략적인 관계를 정의하여 추상적인 모델에서 더 세부적인 모델간의 일관성을 유지하도록 하는 기법이 종종 사용된다. 아키텍처와 상세 설계 사이에 중간모델을 도입하는 것은 객체지향 개발에서의 이런 기법과 흡사한데, 이를 통해 개발자들이 두 산출물의 간격을 줄임으로 자연스럽게 상세설계가 이루어지도록 한다.

중간모델 도입 기법은 그림 2에서 보여주듯이 세가지 주요한 단계들로 이루어진다.

단계 1에서는 C&C 아키텍처에 속하는 컴포넌트, 커넥터, 포트 그리고 역할과 같은 주요 아키텍처 요소들을 각각 객체들과 그들의 관계들로 구성된다. C&C 아키텍처를 구현 요소들로 맵핑하는 것이 이 단계의 주요 활동이다.

이 활동에서는 식별된 아키텍처 요소들을 바탕으로 어떻게 그들이 실현화 될 수 있는지를 고려한다. 먼저 아키텍처 요소들이 가지고 있는 속성이나 연관된 제약사항 등을 바탕으로 특징들을 파악한다. 다음으로 구현요소들의 특징을 파악한다. 마지막으로 가장 효과적으로 아키텍처 요소들을 실현화 할 수 있는 구현요소들을 맵핑시킨다. 하나의 커넥터는 여러 가지로 구현될 수 있다. 예를 들어 하나의 문자열을 한 클래스로부터 다른 클래스로 전송을 하는 경우에는 문자열 버퍼가 사용될 수 있고, 복잡한 데이타 타입들을 하나의 프로세스에서 다른 프로세스로 전송할 경우에는 소켓 통신 또는 미들웨어가 사용이 될 수 있다. 커넥터와 컴포넌트의 구현방법에 대한 선택에 따라 포트의 구현도 달라질 수 있다.

표 1 C&C 아키텍처와 객체지향 설계 단계의 입력물과 산출물

	입력물	산출물
C&C 아키텍처	기능적 요구사항, 비기능적 요구사항	컴포넌트, 커넥터, 포트, 제약사항, 아키텍처 패턴, 수정된 품질속성
객체지향 설계 (클래스다이어그램)	객체, 관계, 기능적 요구사항, 비기능적 요구사항	객체(클래스), 정적인 관계(연관, 병합, 포함, 상속)

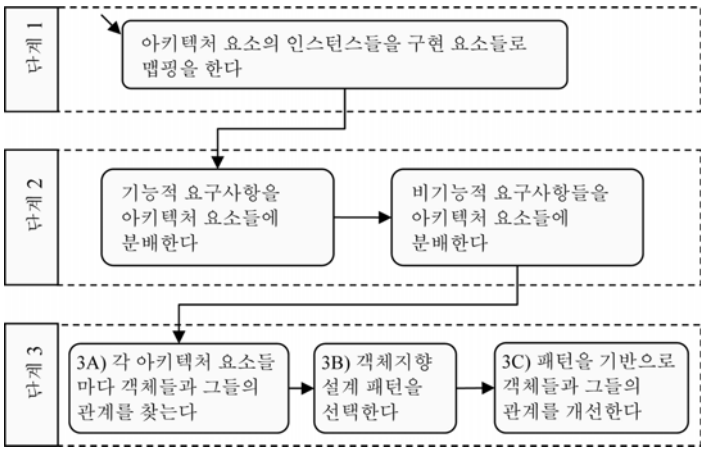

그림 2 중간모델 도입 기법의 절차

예를 들어 미들웨어의 사용은 포트들이 미들웨어에서 메시지 전송자 또는 메시지 수신자가 되도록 요구된다.

각 아키텍처 요소에 대한 구현 요소들이 일단 결정되면 단계 2에서는 기능적 요구사항들과 비기능적 요구사항들이 아키텍처 요소들로 배분된다. 비기능적 요구사항들을 아키텍처 요소들로 배분하는 것은 비기능적 요구사항들이 표현하는 품질속성들이 명시적 또는 암묵적으로 아키텍처 요소들에 의해 지원이 될 수 있기 때문에 기능적 요구사항의 경우처럼 배분이 간단하지 않다. 명시적인 경우 특정 컴포넌트나 커넥터 등의 아키텍처 요소들이 특정 품질요소를 만족하기 위해서 도입이 되거나 추가적인 기능을 가지는 경우이며, 암묵적인 경우에는 여러 아키텍처 요소들의 조합 혹은 이들이 형성하는 구조로 인해서 품질속성이 만족이 되는 경우이다. C&C 아키텍처의 설계 시에 아키텍처적으로 주요한 기능요구사항, 품질속성, 제약사항 그리고 이를 만족시키기 위한 아키텍처 드라이버등을 바탕으로 품질속성의 우선순위를 바탕으로 구조를 결정한다. 만족하지 못한 품질속성은 다음 분배를 통해서 순차적으로 만족시켜 나가는 Attribute Driven Design Method (ADD)를 활용할 수 있다[14]. 아키텍처 설계 시에는 아키텍처적으로 중요한 시나리오를 바탕으로 기능을 분배하게 되고 이 단계에서는 좀 더 구체적인 시나리오나 아직 적용하지 않은 시나리오를 바탕으로 기능요구사항들이 배분되고 검증된다.

단계 3은 세가지 하위 단계들로 구성된다. 첫 번째 하위단계에서는 각 기능적 요구사항들을 만족하기 위한 객체들이 식별된다. 물론 이 접근법의 기본사항에 따라 객체들을 각 아키텍처 요소의 범주에서 식별하는 것만으로 충분하다. 이와 동시에 객체들간에 간략한 관계가 고려된다. 두 번째 하위단계에서는 객체지향 디자인 패턴들이 고려된다. 디자인 패턴들의 적용 없이 객체지향 설계를 하는 것이 어렵기 때문에 이 단계가 필요하다. 물론 이러한 패턴들 없이도 클래스다이어그램을 설계할 수 있지만, 다이어그램의 규모가 점점 커지면 클래스 정렬 규칙의 역할을 하는 패턴들의 적용이 필수적이 된다. 결국 성급히 문제에 대한 해답을 만들기 전에 미리 검증된 디자인 패턴을 고려하는 것이 늦게 고려하는 것보다 더 좋은 설계를 만들어 낸다.

적절한 디자인 패턴을 선택하기 위해서는 각 아키텍처 요소들이 품질속성들을 제약사항들을 준수하면서 이를 구현 가능한 기능적 요구사항으로 변경을 한다. 또한 공통성과 가변성을 식별하여 공통성 부분을 인터페이스 또는 추상 클래스를 사용하고 가변성 부분을 상속 또는 합성(Composition)을 사용하여 추가적인 변경이나 확장이 최소한의 비용으로 구축이 될 수 있는 디자인 패턴을 선택하게 된다. 디자인패턴은 C&C 아키텍처의 설계 결과를 통해 반영된 품질속성들을 제약사항과 구조를 효과적으로 객체지향 설계에 반영하는데 도움을 준다.

품질속성과 요구사항들을 고려할 때 예제에서 제시된 과제에서 컴포넌트들에 적용할 수 있는 패턴들을 표 2에 나열하였다. 디자인 패턴은 반복적으로 나타나는 비슷한 문제들에 대한 일반적인 해답을 제공한다[15]. 표 2에서는 Metadirectory 시스템에 적용되는 디자인패턴들을 보여준다.

마지막 세 번째 단계에서는 식별된 객체들과 그들의 관계들이 선택된 디자인 패턴들에 의해 재정립된다. 디자인 패턴들에 의해 기본적인 객체들의 메소드 호출 순서들이 결정되고 기본적인 그들의 관계가 재정립 된다.

중간모델 도입 기법에서는 비록 순차다이어그램도 포함이 될 수 있으나 이 논문에서는 클래스다이어그램만을 포함한다. 비록 순차다이어그램이 이벤트 발생에 대

표 2 디자인 패턴

디자인 패턴	목적
Command	메소드를 객체로 포장함으로써 다른 객체들 또는 메소드들에 전달이 될 수 있게 함.
Template	일반적인 알고리즘을 기본 클래스로 정의하고 이를 확장하여 상황에 맞게 완성시킬 수 있게 함으로써 알고리즘을 재사용 가능하게 함.
Façade	복잡한 객체들과 그들간의 관계로 구성되는 복잡한 인터페이스를 간략한 인터페이스로 접근하게 만듦.
Observer	객체의 상태변화를 다른 객체들에게 효과적으로 전달 가능하게 하고 이벤트 기반의 상호작용에 기반이 됨.
Singleton	특정 타입의 객체를 오직 하나만 생성함.
Adapter	원하는 기능을 가지고 있지만 인터페이스가 다를 경우 이를 변경하여 사용이 가능하게 함.

한 순서들과 객체들간의 메소드 호출에 대한 더 세부적인 정보를 줄 수도 있으나, 예제를 기술하기에는 클래스 다이어그램만으로 충분하기 때문이다.

5. 중간모델 도입 기법의 적용과 검증

이 절에서는 중간모델도입 기법을 메타디렉터리 시스템에 적용한다. 제5.1절에서는 하나의 아키텍처 요소에 대하여 중간모델도입 기법의 세 단계들을 차례로 적용하는 과정을 보인다. 제5.2절에서는 이 방법으로 도출한 최종세부설계를 보여주고 제5.3절에서는 아키텍처구조와 지원되는 품질속성이 최종세부설계에서 보존 지원됨을 검증한다.

5.1 중간모델 도입

Metadirectory 시스템의 많은 서브 시스템 중에 하나의 서브시스템은 그림 3과 같이 Adapter Manager(AM) 컴포넌트와 그의 커넥터로 구성되어 있으며, 포트들은 포트들을 위한 두 개의 프로토콜을 활용하고 있다. 또한 그 기능이 복잡하여 아키텍처와 상세설계간의 일관성 결여가 발생하기 쉬운 대상이며 중간모델 도입 기법을 적용함으로써 체계적으로 객체지향 설계를 도출하고자 한다.

단계 1) 아키텍처 요소의 인스턴스들을 구현 요소들로 맵핑을 한다.

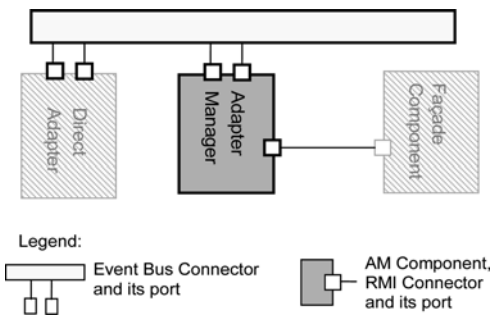

그림 3 Adapter Manager 컴포넌트와 연결된 커넥터와 포트

표 3에서는 아키텍처 요소들과 맵핑이 가능한 구현 요소들 그리고 각각의 특징들을 보여준다. AM 컴포넌트는 Java Thread또는 Java Application으로 구현이 가능하나, 다수의 Director Adapter들과 상호작용을 안정적으로 하기 위하여 보다 좋은 성능이 요구되기 때문에 Java Thread가 구현 요소로 선택이 되었다. Event Bus 커넥터의 구현요소로 Java Message Service (JMS)가 채택되었고 RMI 커넥터의 구현요소로 Remote Method Invocation(RMI)가 기본적인 클라이언트-서버 커넥터에 사용되었다. 각 커넥터의 포트들은 커넥터의 구현요소가 정해짐에 따라 자동적으로 선택이 되었다.

표 3 아키텍처 요소들과 구현 요소들

아키텍처 요소들의 인스턴스들	인스턴스들의 주요 특징	구현 요소들	특징
AM 컴포넌트	Director Adapter와 비동기적 통신이 요구됨. 높은 성능을 바탕으로 다수의 DA의 통신이 지원되어야 함	Java Thread	공유 변수에 접근 이 가능함, IPC에 비하여 효율적인 성능을 가짐
		Java Application	프로세스마다 독립적인 메모리가 요구됨, IPC를 통해 프로세스간에 통신이 이루어짐
Event Bus Connector	Event의 Broadcast	JMS	Java 메시지 서비스로 메시지 큐를 사용하여 이기종 시스템간의 연결에 유용함
RMI Connector	클라이언트 서버의 연결	RMI	분산 객체들간의 메소드를 호출, 복잡한 프로토콜의 구현없이 사용
Event Bus Receiver Port	커넥터에 종속적임	JMS Listener	JMS Listener
Event Bus Sender Port	커넥터에 종속적임	JMS Sender	JMS Sender
RMI Receiver Port	커넥터에 종속적임	RMI Stub	RMI Stub
RMI Sender Port	커넥터에 종속적임	RMI Proxy	RMI Proxy

단계 2) 기능적 요구사항과 비기능적 요구사항의 아키텍처 요소들로의 배분이 확정된다.

표 4는 AM 컴포넌트의 기능적/비기능적 요구사항을 배분을 보여준다. C&C 아키텍처 설계 시에는 아키텍처적으로 중요한 시나리오들을 바탕으로 구조의 결정과 기능이 배분이 된다. 이 단계에서는 아키텍처 레벨에서 비교적 추상적인 요구사항들을 구체화시키고 아직 적용되지 않은 시나리오들을 바탕으로 배분이 확정된다.

AM 컴포넌트는 Direct Adapter(DA)의 (F1) 제거와 (F2) 삭제 그리고 사용자 명령을 DA에게 전달하는 책임을 가지고 있다. 과제의 소프트웨어 요구사항 명세에 따르면 (Q1) DA는 시작 프로세스 또는 전체 시스템의 중지 등의 추가적인 명령의 실행 없이 시스템에 추가되고 삭제될 수 있어야 한다. 이 기능은 시스템의 중요한 품질요소 중 하나인 확장성(extensibility)을 만족시킨다.

사용자는 AM을 통하여 DA들을 조절할 수 있다. 일단 시작 또는 멈춤 명령이 사용자로부터 발생이 되면 먼저 (F3) 목적지 표시와 함께 AM에게 보내지고 명령을 검증한 후에 (F4) 적합한 DA로 재전송된다. 메시지를 전달하는 것 이외에도 (F5) DA의 업무량을 분석하고 (F6) AM 컴포넌트는 이벤트 버스의 큐의 크기를 조절함으로써 (Q2) 메시지의 로드와 관계없이 계속적으로 실행이 될 수 있게 함으로써 시스템의 가용성(availability)를 보장한다.

단계 3A) 각 아키텍처 요소들 마다 객체들과 그들의 관계를 찾는다.

구현 요소들을 아키텍처 요소들에 맵핑하고, 기능적 요구사항과 비기능적 요구사항을 각 요소들로 배분한 후에는, 주요 객체들이 식별되고 간략한 객체 리스트와 비기능적 명세들을 고려하여 적용 가능한 디자인 패턴들 중에 가장 적합한 것을 선택한다. 마지막으로 선택된 디자인 패턴에서 제시된 구조를 기반으로 보완적인 객체들이 추가가능하고 모든 객체들이 정렬되고 관계가 정립이 되어야 한다.

표 5는 각 기능적 요구사항을 위하여 필요한 객체들을 보여준다. 요구사항으로부터 객체들을 산출하는 것이 객체지향 설계의 첫 번째 단계이다. AM 컴포넌트가 수행해야 하는 주요 기능적 요구사항들을 각 객체들로 맵핑한다. 이러한 맵핑은 결합도(cohesion)를 높이고 의존도(coupling)를 낮추는 방향으로 책임을 객체에 분배하게 된다. DA의 삭제와 추가는 높은 결합도를 가지므로 DA_handler로 책임을 할당한다. 컴포넌트의 명령을 확인하는 절차는 명령형식의 변경에 따라 추후 변경될 가능성이 높기 때문에, 변경시 영향을 최소화하기 위해서 새로운 객체인 Command_verifier로 책임을 할당한다. DA들로 메시지를 전달하는 것은 동시에 일어날 수 있는 여러 문제들과 비동기적 요소들을 고려해야 하는 복잡한 책임을 가지고 있기 때문에 Command_sender라는 새로운 객체로 책임을 할당한다. DA의 업무량 분석과 이벤트 버스의 큐 사이즈 조절은 밀접한 연관성을 가지기 때문에 하나의 객체로 책임이 할당될 수 있으나 추후 복잡한 큐 조절 알고리즘과 다른 로드 분산 방법의 확장을 고려해서 Load_checker와 Queue_controller로 책임이 할당된다. 객체들은 개략적이라서 후에 디자인 패턴 등을 고려하여 더 작은 객체들로 분리될 수 있거나 구조적 완성도를 위해 추가적인 객체들이 필요할 수 있다.

산출된 각 객체들은 공통 변수들을 공유하거나 다른 객체에 메소드를 호출함으로써 상호작용할 수 있다. 상세 설계에서는 객체들의 관계로 이러한 상호작용이 표

표 4 AM 컴포넌트를 위한 기능적/비기능적 요구사항의 배분결과

기능적 요구사항	비기능적 요구사항
F1. DA들을 제거한다. F2. DA들을 추가한다. F3. façade 컴포넌트로부터 명령을 검증한다. F4. 목표된 DA들로 메시지를 전달한다. F5. DA의 업무량을 분석한다. F6. 업무량에 맞추어 이벤트 버스의 큐 사이즈를 조절한다.	Q1. 확장성: A. 시스템은 DA의 추가 및 삭제가 쉽게 재시작 또는 재배치 없이 가능하도록 한다. B. 이 컴포넌트에 새로운 기능이 쉽게 추가되도록 한다. Q2. 가용성: C. 메시지의 로드와 관계없이 계속적으로 실행되어야 한다.

표 5 기능적 요구사항을 만족하기 위해 필요한 객체들

요구사항들	필요한 객체들
DA들을 삭제한다 DA들을 추가한다	DA_handler
Façade 컴포넌트로부터 명령을 확인한다	Command_verifier
목표된 DA들로 메시지를 전달한다	Command_sender
DA의 업무량을 분석한다	Load_checker
업무량에 맞추어 이벤트 버스의 큐 사이즈를 조절한다	Queue_controller

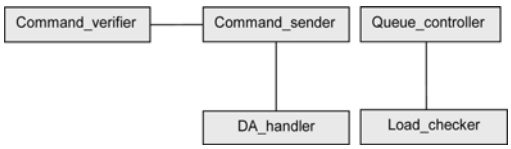

그림 4 주요 객체간의 관계

현되기 때문에 관계를 특징짓는 것이 중요하다.

단계 3B) 객체지향 설계 패턴을 선택한다.

일단 객체들과 그들의 관계들로 구성된 전체적인 구조가 결정이 되면 비기능적 요구사항들을 반영하기 위하여 디자인 패턴들이 선택되고 전체적인 시스템의 구조가 개선된다. 예를 들어 Factory 패턴을 사용함으로써 Adapter proxy가 AM에서 쉽게 생성되고 소멸될 수 있었다. 비록 확장성을 지원하는 주요 부분은 Publish & Subscribe의 아키텍처 스타일을 사용하는 것이지만 이 패턴을 사용하는 것 또한 품질속성을 만족시키는데 많은 역할을 했다. Adapter 패턴을 이용함으로써 기존 클래스의 변경 없이 새로운 클래스가 원하는 인터페이스에 맞도록 사용이 될 수 있게 하고 Template 패턴은 일반적인 알고리즘을 내포하는 템플릿 객체를 이용하여 재사용성을 높이고 또한 각 요구사항에 맞게 알고리즘을 명세화할 수 있도록 한다. 마지막으로 Command 패턴은 명령을 하나의 객체로 취급하게 함으로써 명령의 취소와 그룹화가 쉽게 수행이 되도록 한다. 표 6은 선택된 패턴들을 보여준다.

단계 3C) 패턴을 기반으로 객체들과 그들의 관계를 개선한다.

그림 5와 표 6의 패턴들은 중간모델 도입 기법의 결과 산출물이다. 디자인 패턴을 모델에 적용하기 위하여 요구되는 객체들이 그림 5에 진한 회색으로 추가가 되었다. 추가된 객체 중에 Config_manager와 Adapter_factory는 Factory 패턴을 적용하기 위해 추가되었고 Adapter_factory와 Adapter_for_DA_proxy는 Adapter 패턴을 적용하기 위해 추가되었다. 마지막으로 DA_proxy는 Proxy 패턴을 적용하기 위해서 추가되었다. 또한 이렇게 추가된 객체들 간에도 서로 연관관계를 맺으며 상호보완적인 관계를 맺을 수 있고 이러한 변화로 인해 객체들간의 관계가 재정립이 된다. 이 단계에서는 중간모델 도입 기법은 상속이나 포함관계를 사용하지 않은 관계들에만 국한이 되어있기 때문에 패턴에 대한 명백한 특징이 나타나지 않는다. 그러나 선택된 디자인 패턴들을 고려하여 차후 설계와 일관성이 있도록, 패턴, 기능적 요구사항 그리고 비기능적 요구사항에 필요한 중요 객체들을 찾아낸다는 점에서 중요한 의미를 가진다.

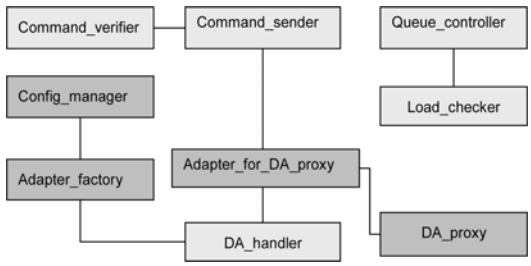

그림 5 AM 컴포넌트를 위한 중간 모델

표 6 비기능적 요구사항을 기반으로 선택된 디자인 패턴
(A와 B는 표 4에서 제시한 확장성의 타입들이다)

요구사항	디자인 패턴
확장성 A*	Factory
확장성 B**	Factory, Adapter, Template, Singleton
사용성	Command

5.2 최종 객체지향 설계

이전 절에서는 AM 컴포넌트만을 고려한 중간단계 모델을 제시하였다. 다른 아키텍처 요소들(예를 들어 커넥터 그리고 포트들)도 비슷한 방법으로 적합한 중간단계 모델들을 구축할 수 있다. 이러한 모델들을 바탕으로 포트들을 포함하는 AM 컴포넌트도 그림 6과 같이 기술될 수 있다.

커넥터를 위한 포트를 컴포넌트가 어떻게 수용하고 있는지를 보여주기 위해서 원 클래스 다이어그램에 있던 명령어 처리와 적재 조절은 생략이 되었고 오직 포트에 연결된 Adapter proxy의 생성만이 강조되었다. 이 다이어그램에서는 사선으로 채워진 클래스들은 아키텍처의 포트 부분을, 일반 클래스들은 컴포넌트를 나타낸다.

그렇다면 C&C 아키텍처로부터 직접적으로 도출한 클래스다이어그램과 중간단계 모델로부터 도출한 것의 차이가 무엇인가? 가장 중요한 차이는 중간모델 도입 기법으로부터 도출된 클래스들은 동일한 방법으로 관리가 된다는 것이다. 즉 요구사항이 변경되어 아키텍처의 어떤 부분이 수정이 되면 앞의 프로세스에서 만들어진 테이블을 참고하여 해당 객체지향 설계 부분을 수정하면 된다. 이는 품질 요소의 변경에도 동일하게 적용될 수 있다. 결국 기능/비기능 요구사항들이 일관성 있게 상세설계에도 관리 될 수 있다.

5.3 검증

중간단계 모델 없이 시스템을 개발하는 것은 가능하다. 하지만 일관성, 역공학, 유지 그리고 개발관점에서 C&C 아키텍처를 객체지향 설계로 전환하는 명확한 프로세스가 중요한 역할을 할 수 있다. 이 절에서는 중간단계의 모델 도입의 효율성이 두 가지 관점에서 제시될 것이다. 하나는 “아키텍처가 일관성 있게 유지되었는

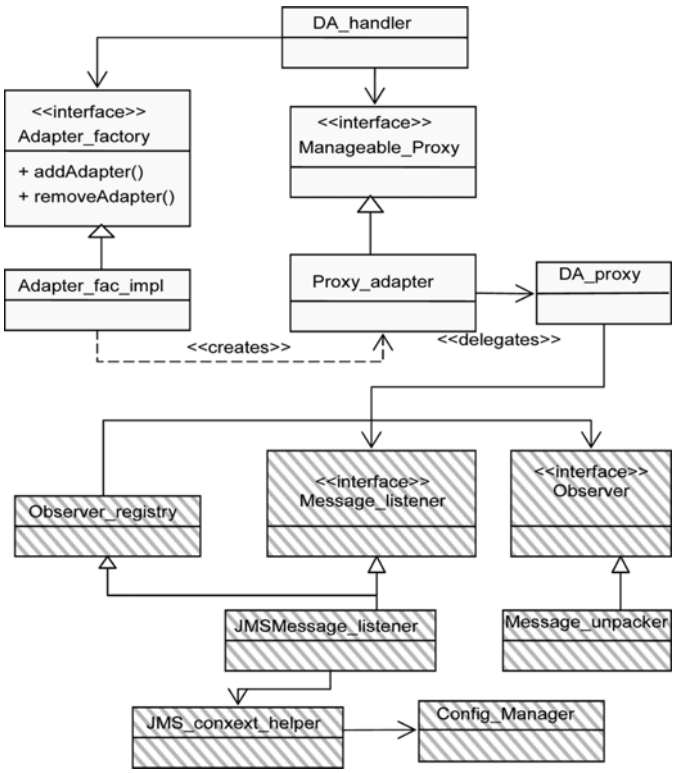

그림 6 AM 컴포넌트의 간략한 최종 클래스다이어그램

가?"이고 다른 하나는 "중간모델 도입 기법을 수행하는 과정에서 품질속성들이 그대로 유지가 되었는가? 이다.

아키텍처 구조의 보전

중간모델 도입 기법은 각 아키텍처 요소마다 수행이 된다. 첫 번째 중간모델 도입과정은 프로그래밍의 구조들을 C&C 아키텍처 요소들로 맵핑하는 것이다. 남은 과정 부분은 각 프로그래밍의 구조들마다 다른 요소들을 고려하지 않고 독자적으로 수행된다. 따라서 중간단계 모델의 산출물들은 객체 또는 클래스들과 아키텍처 요소들간에 관계를 정의하고 유지함으로써 전반적인 아키텍처의 구조들은 객체지향 설계의 구조들에 일관성 있게 유지될 수 있다.

품질속성들의 유지

많은 연구자들이 아키텍처의 중요성의 하나가 시스템에 필요한 품질속성들을 아키텍처가 담을 수 있기 때문이라고 말한다. 하지만 문맥 수준의 품질속성들은 종종 서브시스템 또는 객체의 속성들이 그들을 지원할 때에만 그 의미가 유지된다. 표 7은 품질속성들이 어떻게 모델링 과정 동안에 유지될 수 있는지를 기술한다. 첫 번째 단계에서 아키텍처 요소들을 이들의 구조를 유지하고 품질속성을 실현시킬 수 있을 구현요소들로 맵핑을 하고 두 번째 단계의 모델링은 비기능적 요구사항을 각 아키텍처 요소들로 분배하고 그 과정에서 품질속성은 각 요소들 또는 요소들의 구조로 명백하게 할당된다. 마지막 세 번째 단계에서 품질속성들을 지원할 수 있는 디자인패턴들을 선택을 함으로써 아키텍처의 명세부터 세부 객체지향 설계까지 유지되게 된다.

이러한 방법으로 아키텍처와 품질속성들은 중간모델

표 7 IM에서 지원하는 품질속성들

아키텍처 요소	구현 요소	적용된 디자인 패턴	지원된 품질속성
컴포넌트	Java Process	Factory, Adapter, Template, Command	Extensibility, Usability
포트	JMS sender	Observer	Integrity
	JMS receiver	Observer	
커넥터	JMS	Singleton	Usability
	RMI	Singleton	

도입 기법을 통해 컴포넌트와 커넥터 아키텍처부터 아키텍처 설계까지 유지가 된다.

제2절에서 소개한 Metadirectory 과제 수행 시에 발생한 문제들이 해결이 되었는지를 살펴보자. 첫 번째 문제인 P1은 다리의 역할을 하는 중간단계 모델이 C&C 아키텍처를 입력으로 받고 클래스다이어그램의 입력으로 쓸 수 있는 객체들과 그들간의 관계를 산출함으로써 해결이 되었다. P2는 객체 리스트와 디자인 패턴들이 얼마나 많은 객체들이 필요하고 아키텍처를 실현하기 위해서 어떠한 관계가 필요한지를 정의하기 때문에 구현기반이 제공 되었다고 말할 수 있다. 이 구현기반 없이는 세부 설계들은 일관성을 잃을 수 있다. P3은 아키텍처 스타일에 따라서 어떤 특정한 설계 패턴이 적용되는 것이 아니라, 그 스타일을 구성하는 요소들에 따라 필요한 컴포넌트를 만들고 맵핑시키는 프로세스를 이용함으로써 일관성 있는 개발이 가능하게 되었다. P4는 중간단계 모델을 기반으로 비기능적 속성에 대한 역추적을 함으로써 역시 해결이 되었다.

4. 결 론

아키텍처 요소를 미들웨어 등의 구현 개체들에 맵핑하는 것이 중간모델 도입 기법의 기본이다. 이 과정은 시스템이 객체지향 방식으로 구현될 때에도 아키텍처 구조가 유지되는 것을 보장한다. 중간모델 도입 기법은 이 기본 과정에 두 가지 중요한 활동을 더한다. 첫 번째는 요구되는 주요한 객체들을 뽑아내고 그들간의 관계를 결정하는 것이며 두 가지 모두 객체지향 설계에 중요한 입력이 된다. 두 번째는 디자인 패턴에 관한 것이다. 산출된 객체들과 비기능적 요구사항을 기반으로 디자인 패턴들을 선택하고 모델에 적용한다. 이 과정은 품질속성들이 객체지향 설계에 지속적으로 유지되게 하여준다. 나아가 객체들의 리스트와 그들의 관계를 만듦으로써 상세한 정도의 수준과 구현 방법을 제한하여 객체지향 설계가 일관성 있게 구축될 수 있었다.

품질속성 및 아키텍처 요소들에 맞게 디자인 패턴을 선택하는 것은 쉬운 일이 아니다. 이 논문에서는 경험적 접근법을 이용하여 개발자가 수용하고 구체화 할 수 있는 디자인 패턴을 선택하였다. 일반적인 디자인 패턴들은 서로간에 매우 복잡한 관계들을 가질 수 있어서 적절한 조화를 통해 더 나은 품질속성을 반영할 수 도 있고 역효과를 가질 수도 있다. 때문에 이상적으로는 이를 고려한 체계적인 접근방법이 필요하다. 더 체계적인 프로세스를 위해서 디자인 패턴을 선택하는 정형화된 과정의 연구는 향후 과제로 남아있다.

참 고 문 헌

[1] D. Garlan and M. Shaw, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996.

[2] Schmerl and D. Garlan, "AcmeStudio: Supporting Style-Centered Architecture Development," In *Proc.* of the 26th International Conference on Software Engineering, Edinburgh, Scotland, May 23-28, 2004.

[3] D. Garlan, R. Allen, and J. Ockerbloom, "Exploiting Style in Architectural Design Environments," In *Proc.* of SIGSOFT '94: Foundations of Software Eng., pp. 175-188, Dec. 1994.

[4] N. L. Kerth and W. Cunningham, "Using patterns to improve our architectural vision," Software, IEEE, vol. 14, issue 1, pp. 53-59, Jan.-Feb. 1997.

[5] N. Medvidovic and R. N. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE Transactions on Software Engineering*, vol. 26, issue 1, pp. 70-93, Jan. 2000.

[6] M. Bjorkander and Cris Kobryn, "Architecting Systems with UML 2.0," IEEE Software, 2003.

[7] S. Cheng and D. Garlan, "Mapping Architectural Concepts to UML-RT," Int'l Conf on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001), Monte Carlo Resort, Las Vegas, Nevada, USA, Jun. 2001.

[8] Object Management Group, CORBA 3.0 New Components Chapters, ptc/99-10-04, Oct. 1999.

[9] Microsoft, The global XML Web services architecture GXA; available at http://msdn.microsoft.com/webservices/understanding/gxa/default.aspx

[10] N. Medvidovic, "On the Role of Middleware in Architecture-Based Software Development," ACM 1-58113-556-4/02/0700, 2002.

[11] J. Aldrich, C. Chambers, and D. Notkin, "ArchJava: Connecting Software Architecture to Implementation," In *Proc.* of ICSE 2002, May 2002.

[12] A.H., Eden and R. Kazman, "Architecture, design, implementation," In *Proc.* of the 25th International Conference on Software Engineering, pp. 149-159, May 2003.

[13] J. Ivers, P. Clements, D. Garlan, R. Nord, B. Schmerl, & J. Silva, *Documenting Component and Connector Views with UML 2.0*, Technical Report CMU/SEI-2004-TR-008 ESC-TR-2004-008.

[14] F. Bachmann, L. Bass, G. Chastek, P. Donohoe, & F. Peruzzi, *The Architecture Based Design Method*, CMU/SEI-2000-TR-001 ESC-TR-2000-001.

[15] E. Gamma, R. Helm, R.E. Johnson, and J. Vlissides, *Design Patterns*, Addison-Wesley, 1994.

박 형 일

1999년 포항공과대학교 이학(화학사). 2004년 미국 Carnegie-Mellon University 전산학(소프트웨어공학석사). 2004년 한국 정보통신대학교 공학(소프트웨어공학석사). 2004년~2005년 (주)티맥스소프트 선임 컨설턴트. 2005년~현재 A.T Kearney Korea LLC 컨설턴트. 관심분야는 BPM, 소프트웨어 구조

강 성 원

1982년 서울대학교 사회과학대학(정치학사). 1989년 Univ. of Iowa 전산학(전산학석사). 1992년 Univ. of Iowa 전산학(전산학박사). 1993년~2001년 한국통신 연구개발본부 선임연구원. 1995년~1996년 미국 국립표준기술연구소(NIST) 객원연구원. 2001년~현재 한국정보통신대학교 부교수. 2003년~현재 미국 Carnegie-Mellon University 소프트웨어공학석사과 겸임교수. 관심분야는 소프트웨어구조, 소프트웨어 시험, 형식기법

최 윤 석

2001년 인하대학교 공과(전산학사). 2001년~2002년 삼보컴퓨터 기술연구소 소프트웨어개발팀. 2005년 한국 정보통신대학교 공학(소프트웨어공학석사). 2005년~2006년 한국정보통신대학교 소프트웨어 공학 연구소 연구원. 2006년~현재 박사과정. 관심분야는 소프트웨어 프로세스, BPM, 소프트웨어 구조

이 단 형

1971년 서울대학교 공과대학(공학사). 1983년 Arthur D. Little(경영과학석사). 1992년 Virginia Commonwealth Univ.(정보시스템박사). 1972년~1995년 KIST/시스템공학연구소 선임연구부장. 1996년~1998년 ETRI 소프트웨어공학 연구부장 1991년~현재 ISO/IEC JTC1/SC7 WG4 의장(SW 자동화 도구 및 방법론). 2003년~현재 한국정보통신대학교 교수 2003년~현재 미국 Carnegie-Mellon University 소프트웨어공학석사과정 겸임교수. 관심분야는 소프트웨어프로세스, 요구공학, 프로덕트라인, 도구 및 방법론