# 소프트웨어 구조스타일의 정형화를 통한 조립형 구조패턴의 정의

궁 상 환†


## 요 약

소프트웨어 재사용의 문제는 두 가지 측면에서 검토해 볼 수 있는 데, 하나는 이미 개발이 완료된 소프트웨어 컴포넌트를 조합하여 재사용 하는 방법이고, 다른 한 가지는 소프트웨어들이 공통적으로 갖는 구조적인 패턴을 정의해서 재사용 하는 방법이다. 물론 이 두 가지가 소프트웨어의 구조설계에 중요한 방법이긴 하지만, 이 두 방식의 차이점은 소프트웨어 컴포넌트는 프로그램 컨텐츠가 채워진 빌딩블록들을 재사용 하여 소프트웨어를 쉽게 구성하는 것인 반면, 소프트웨어 구조패턴은 빌딩블록의 내용보다는 빌딩블록의 틀, 즉 빌딩블록간의 연결을 통한 구조화에 더 관심이 있는 것이라고 하겠다. 논문은 후자의 경우와 같이 소프트웨어 응용에서 많이 발견되는 구조적인 소프트웨어 패턴을 분석하고 체계적으로 정의하여, 이 패턴들이 소프트웨어 구조 설계 시 유용하게 활용될 수 있도록 하는 데 목적을 두고 있다. 특히, 이제까지 소프트웨어의 구조적인 패턴을 설명하는 데 중요한 모델이 되어 왔던 소프트웨어 구조 스타일을 분석하고 정형화하여 다양한 응용의 소프트웨어 구조설계에 활용될 패턴형 컴포넌트를 정의하고자 한다.


# Defining of Architectural Patterns through Formalization of Architectural Styles

Sang Hwan Kung†


## ABSTRACT

The problem of software reuse is dealt in two approaches. One is to build a new software by composing of the built-in components, and the other is to reuse architectural patterns that most of software system is generally composed of. Although the two approaches are important in design of software architecture, we could find outstanding difference in what kind of building blocks they use. The component based software design makes uses of building blocks whose contents are filled in by someone, on the other hand, the architectural pattern based software design is not interested in the contents of building blocks, but in the framework for building blocks including relationship of the building blocks. The paper purposes to find architectural patterns which are commonly found in diverse applications and help software architects reuse them in the software design process. We refine the architectural styles which is the well-known concept for software architecture design, and refine them as architectural components or templates which can be parts of software architecture.


**키워드 : 소프트웨어 구조(Software Architecture), 구조 스타일(Architectural Styles), 구조 패턴(Architectural Patterns), 디자인 패턴(Design Patterns)**

## 1. 서 론

우리는 소프트웨어 설계를 수행할 때, 특정한 응용에서 요구하는 기능(responsibility)에 대해 통상 두 가지 접근방식을 찾게 된다. 하나는 요구 기능을 소프트웨어로 구조화하는 데 적합한 구조 패턴을 매핑 하는 작업이며, 다른 하나는 바로 그 기능이 구현물로 되어 있는 재사용 컴포넌트를 찾는 일이다. 실제 이 두 방법은 응용에 따라 차이는 있으나 상호 보완적으로 활용된다. 예를 들어, 비즈니스 로직을 구현하고자 하는 경우에는 Java Beans나 .com 등을 이용하여 이미 상용화나 개발이 완료된 컴포넌트들을 쉽게 찾을 수 있기 때문에 컴포넌트의 활용이 구조 패턴의 활용보다 더 중요하다. 그러나 게임이나 영상회의, VOD와 같은 응용의 경우는 요구되는 기능을 구현하기 위한 구조 패턴의 결정이 용이하지 않을 뿐 만 아니라 소프트웨어 품질에 미치는 영향이 크기 때문에 구조 패턴의 활용이 더욱 중요할 수 있다.

본 논문은 컴포넌트의 재사용 측면이 아니라 소프트웨어의 구조 패턴의 활용을 지원하기 위한 방법론을 제시하고 있다. 즉, 소프트웨어 조직적인(structural) 구성을 분석하여

† 정 회 원 : 천안대학교 정보통신학부 교수
논문접수 : 2002년 4월 3일, 심사완료 : 2002년 6월 24일

다양한 응용에서 나타나는 구조 패턴을 발견하고 정의하는 것을 연구의 목적으로 하고 있다. 이를 위해 소프트웨어의 기본적인 자원요소인 프로세스나 버퍼 그리고 자료 및 제어의 흐름을 이용하여 공통적인 구조 패턴의 유형을 다이어그램으로 설명한다. 또한 각 패턴의 적용 환경과 특징을 설명하여 상황에 따른 선택과 활용이 용이하도록 한다. 아울러 소프트웨어 설계단계에서 구조 패턴이 활용되는 절차에 대한 소개를 연구의 범위에 포함하여 언급하고자 한다.

논문의 구성은 제2장에서 소프트웨어 구조설계를 지원하는 구조패턴 관련연구에 대해 구조 패턴과 구조 스타일, 디자인 패턴을 중심으로 설명한다. 제3장에서는 구조 패턴의 유형을 특징 및 사례와 함께 설명하고 구조 패턴간의 관련성을 정의한다. 제4장은 구조 패턴의 활용 예제와 활용 효과를 제시하며, 제5장의 결론에서 연구의 미비점과 향후의 연구방향을 기술한다.

## 2. 관련 연구

### 2.1 소프트웨어 구조 패턴

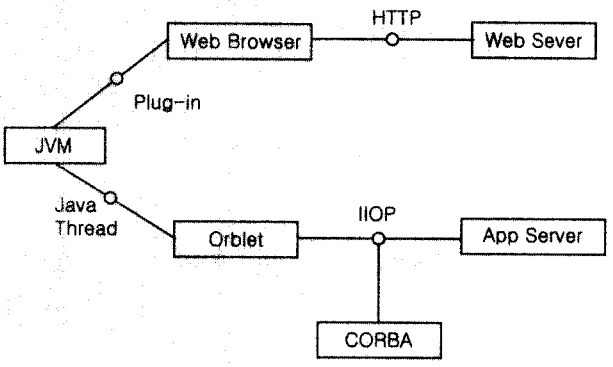
소프트웨어 구조(architecture)는 소프트웨어를 구성하는 컴포넌트와 컴포넌트들의 속성 그리고 컴포넌트간의 관계를 포함하는 시스템의 구조 또는 구조들의 집합이다[9]. 여기서 소프트웨어 컴포넌트란 특정한 상호작용 규칙과 조합을 위한 표준을 준수하는 경우 특별한 수정 없이 그대로 활용이 가능한 소프트웨어 구성요소이다[5]. 컴포넌트는 미리 규격에 따라 제작된 소프트웨어 요소를 부품과 같이 조립하여 무한정 재사용 할 수 있다는 장점을 준다. 컴포넌트 외에 구조설계 단계에서 소프트웨어 설계자가 활용할 수 있는 재사용 요소에는 구조 패턴(Architectural Patterns)이 있다. 구조 패턴은 소프트웨어 시스템을 위한 기본적인 스키마로 사전에 정의된 서브시스템을 정의하고 그들간의 관계를 정의하기 위한 규칙과 가이드라인을 소개한다[4].

패턴 지향의 소프트웨어 구조 유형에는 구조패턴과, 디자인패턴 그리고 이디엄이 있다. 이 중 구조 패턴은 가장 상위레벨에서의 소프트웨어 구조물간의 관계를 표현한다. 한편 디자인 패턴은 중간 정도 규모의 패턴으로 특정한 프로그래밍 언어에 독립적이다. 이들과 비교할 때 이디엄은 프로그램 언어에 종속적인 가장 낮은 수준의 패턴으로 분류된다.

이 장에서는 컴포넌트간의 상호작용을 중심으로 한 구조 패턴의 최근 연구 및 동향에 관한 현황을 분석하면서 관련된 특징과 문제를 살펴본다.

#### 2.1.1 구조 앙상블

앙상블(Ensemble)은 기술과 제품 그리고 컴포넌트들 사이의 존재하는 상호작용이 어떤 유용한 또는 예측된 총체적 행위를 제공하는 것을 의미한다[8]. COTS(Commcial Off-The-Shelf) 기반의 소프트웨어 구조 앙상블(Architectural Ensemble)은 상용화된 소프트웨어 컴포넌트들이 일정한 상호작용 패턴을 통해 상호 결합하여 소프트웨어 구조를 형성하는 개념이다. 이러한 컴포넌트에는 DBMS나 웹서버와 같은 독립적으로 활용되는 패키지 소프트웨어도 있다. 그리고 인터넷 브라우저 안에 포함된 Java Virtual Machine과 같이 다른 소프트웨어의 구성요소로 활용되는 소규모 컴포넌트도 포함될 수 있다. (그림 1)은 인터넷 환경에서 웹 환경 구축에 필요한 컴포넌트들을 중심으로 앙상블 구조를 형성하는 예를 보여 주고 있다.



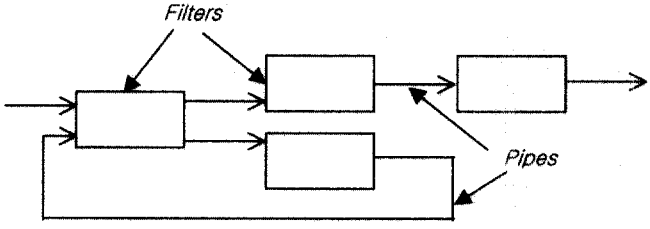

(그림 1) COTS의 구조 앙상블

이와 관련한 또 다른 연구는 IBM의 e-business와 관련한 구조 패턴의 정의를 들 수 있다. 이들은 전자상거래 시스템의 구축의 일반적 유형인 사용자-비즈니스, 비즈니스-비즈니스, 사용자-사용자 등 다양한 시나리오별로 필요한 컴포넌트들과 그들의 관계를 패턴화하여 정의하고 있으며, 구현자들이 경우에 따라 적합한 패턴을 선택하여 활용할 수 있도록 지원한다[6].

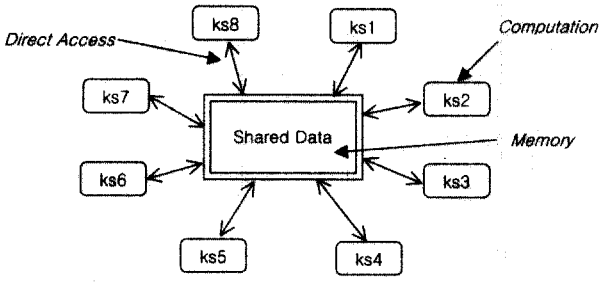
#### 2.1.2 구조 스타일

소프트웨어 시스템은 시스템을 구성하는 조직적인 측면에서 어떠한 스타일을 갖고 있다. 특히 유사한 도메인이나 유사한 환경에서 동작하는 응용은 서로 유사한 구조 스타일(Architectural Styles)을 사용하게 되는 경우가 많다. 각각의 스타일은 어떠한 형태로 구성되어 있는 가를 보여 주기 위하여 공통된 프레임워크를 갖는다. 이 프레임워크에서는 하나의 시스템을 연산/처리 컴포넌트의 집합으로 묘사하며, 이 컴포넌트들 사이는 컨넥터들이 위치하여 이들간의 통신과 조정, 협력을 지원한다. 구조 스타일은 소프트웨어를 구성하는 조직의 패턴에 따라 시스템들의 그룹을 분류하는 것이며 더욱 상세히 묘사한다면 컴포넌트와 컨넥터의 유형에 대한 용어 그리고 이들이 어떻게 결합되는 가에 대한 제약조건의 집합을 정의한다. 또한 설계자가 구성된 부분의 알려진 특성을 분석하여 시스템의 전반적인 속성을 이해하는 데 필요한 semantic model을 제시한다[10].

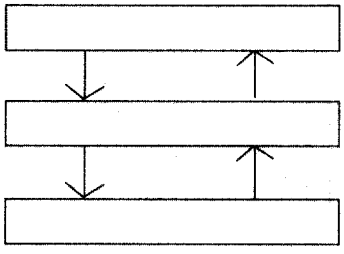
(그림 2-a), (그림 2-b), (그림 2-c)는 구조 스타일의 몇 가지 예를 보여 준다. Pipes-and-filters 스타일은 한 모듈에 입력되는 데이터 스트림을 임의의 조건에 의해 필터링을 해서 다음 모듈로 연결해 주는 형태이며, Blackboards 스타일은 복수의 모듈이 동일한 정보를 메모리를 통해 공유하는 형태이고, Layered 스타일은 통신 프로토콜과 같이 모듈간 상하위 계층구조를 형성하는 구조 스타일이다. 이제까지 구조 스타일의 개념은 보다 개념적이고, 일반적이며, 광범위하다. 즉, 어떤 소프트웨어가 어떠한 스타일에 포함되는 가에 집중하였으며, 이 스타일들이 소프트웨어 구조의 컴포넌트로 활용되기 위해서는 보다 더 많은 분석과 개념의 확장이 필요하다 하겠다.




(그림 2-a) Pipes-and-filters 스타일




(그림 2-b) Blackboards 스타일



(그림 2-c) Layered 스타일

2.1.3 디자인 패턴

디자인 패턴(Design Patterns)은 객체지향 소프트웨어 구조설계 및 세부설계를 지원하여 유연성이 있고 재사용이 가능한 객체지향 소프트웨어를 가능하게 한다. 즉, 여러 클래스들의 조립을 통해 경우마다 처리방법을 직접 코드에 반영하는 hard coding이 아니라, 각각의 패턴이 제공하는 클래스들의 관계(structural framework)를 그대로 활용하도록 하면서 특정 클래스의 수정이 프로그램의 골격을 손상시키는 일이 없도록 하는 soft coding을 가능하게 한다.

디자인 패턴은 구조 스타일과 비교해 볼 때 모두가 소프트웨어의 구조적 패턴을 지원하는 개념을 포함하고 있다는 점은 비슷하나, 어느 정도의 추상화 레벨에서 구조적 패턴을 지원하는 가에는 서로 차이가 있다. 그러나 이 두 방법은 별개의 개념으로보다는 상호 보완적으로 사용할 수 있다. 즉, 구조 스타일은 빌딩 블록을 위한 디자인 요소들의 집합 또는 이러한 빌딩블록을 구성하기 위한 규칙 및 제약뿐 아니라 구조 스타일 내에서 생성되는 설계요소를 분석하고 관리하는 데 필요한 도구를 제공한다. 한편, 디자인 패턴은 특정한 하나 또는 여러 개의 구조 스타일 안의 더 작고 세부적인 문제를 해결하는 문제에 초점을 맞추고 있으므로 설계단계에 맞추어 이 두 방법을 효과적으로 활용할 수가 있는 것이다[2].

2.2 모델 지향 소프트웨어 구조

Object Management Group(OMG)은 기존의 CASE 도구가 최근의 운영체계나 O/S 그리고 클라이언트-서버 구조 환경에 적합하지 않다고 보고, 프로그램언어나 벤더, 미들웨어에 중립적인 소프트웨어 시스템의 명세기법인 MDA (Model Driven Architecture)를 연구하고 있다. MDA는 응용의 목표 플랫폼에 독립적인 방법으로 비즈니스의 기능과 행위를 기술하는 PIM(Platform Independent Model)과 이 PIM을 미들웨어 기술에 매핑한 PSM(Platform Specific Model)을 이용하여 다양한 컴퓨팅 환경에 적합한 소프트웨어의 설계를 지원한다. MDA의 이 두가지 모델은 UML(Unified Modeling Language)을 이용하여 기술되며, 소프트웨어 개발 생명주기의 전과정을 지원하고 있다[7].

2.3 구조명세 언어

Acme는 카네기멜론 대학에서 소프트웨어 구조를 기술하는 언어로 제안된 언어이다. Acme는 컴포넌트와 컨넥터 등 소프트웨어 구조를 형성하는 중요한 요소들의 속성과 관계의 명세를 통해 소프트웨어 구조정의를 위한 다이그래밍 방법이다[9]. Acme와 관련한 최근의 연구에서는 효율적인 구조 명세의 지원 뿐 아니라 구조표현 명세에 대한 교환을 위해 XML을 기반으로 하는 ADML(Architecture Description Markup Language)에 대한 연구가 포함되고 있다[1].

2.4 구조기반 설계

카네기 멜론 대학의 Software Engineering Institute(SEI)에서 연구 중인 구조기반설계(Architecture Based Design)는 Top-down 방식으로 시스템을 분해하여 소프트웨어 구조를 설계하는 과정에서 설계요소(Design Elements)라고 부르는 개념적인 서브시스템 또는 개념적인 컴포넌트들을

도출하게 된다. 이 방법에서 소프트웨어 구조는 각각의 설계요소들을 논리 뷰(Logical View)와 동시 뷰(Concurrency View), 배치 뷰(Deployment View)라는 3가지 view를 통해 표현되며, 반복적인 설계과정을 통해 각각의 view를 개선시킨다[2].

한편 Open Group에서는 정보기술구조의 설계를 지원하는 TOGAF(The Open Group Architectural Framework)의 표준화를 진행 중에 있다. TOGAF의 핵심은 ADM(Architecture Description Method)가 제시하는 7단계의 정보기술구조 개발주기를 통해 정보기술구조를 정의한다. ADM은 구조적인 빌딩블록을 재사용하기 위한 개념으로 ABB(Architectural Building Block)과 SBB(Solution Building Block)을 제시한다. ABB는 어떠한 빌딩블록인가(what)의 관점에서 빌딩블록의 기능적, 기술적인 요구사항을 기술하는 반면, SBB는 빌딩블록의 구현(how)의 관점에서 요구사항을 만족하는 구현물을 의미한다. ADM은 초기 ABB의 설계 이후에 ABB에 활용 가능한 SBB를 선정을 통하여 단계적으로 시스템을 구축하는 방법을 제안하고 있다[11].

## 3. 소프트웨어 구조 패턴의 정의

### 3.1 접근 방법

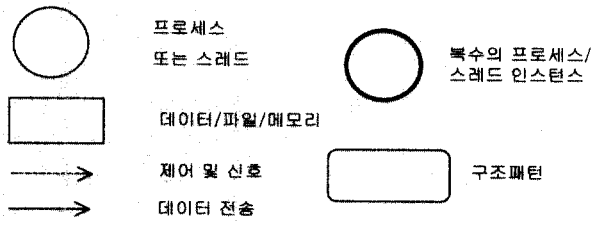
우리가 소프트웨어 구조 스타일을 좀 더 자세히 살펴보면, 구조 스타일에는 Pipes-and-filters나 Blackboards 스타일과 같이 구체적인 스타일도 있지만 Call-and-return-system 스타일처럼 거의 모든 프로그램에서 나타나는 스타일도 있고 Layered 타일과 같은 경우는 Pipes-and-filters 스타일로 분해될 수 있다는 것도 발견할 수 있다.

본 논문의 관점은 소프트웨어 개발자가 어떠한 응용을 만났을 때 그 응용에 대한 소프트웨어 구조를 설계하고, 특히 소프트웨어의 성능과 모듈화를 동시에 고려해야 하는 많은 상황에서 어떠한 구조 스타일의 컴포넌트를 이용해야 할 것인가를 제시하는 문제에 관심을 두고자 한다. 이를 위해 우선 구조 스타일을 분석하고 구체화하여 소프트웨어 구조설계에서 보다 유용한 패턴으로 스타일의 집합을 정의하고자 한다.

소프트웨어 구조 스타일이 소프트웨어가 포함하는 일반적인 구조적인 유형을 제시하였다면 본 연구에서는 보다 구체적인 응용사례의 조사를 통하여 구조 스타일을 분석하고 활용사례를 검토하고자 하는 것이다. 즉, 구조 스타일을 분해하고 결합하여 구조설계에 유용한 새로운 구조적 결합물인 구조 패턴을 정의하여 소프트웨어 구조설계과정에서 부품으로 활용 가능한 구조적 패턴을 개발하고자 한다. 특히, 구조 패턴을 결정하는 중요한 단위로서 디스크와 메모리, 공유 메모리, 파이프, 프로세스, 그리고 스레드 같은 요소들이 어떠한 형태로 소프트웨어를 구성하는 가에 관심을 두고 있다. 이러한 요소간의 데이터의 흐름 뿐 아니라 제어의 흐름을 고려하여 명쾌한 패턴의 정의가 되도록 하고자 한다.

### 3.2 구조 패턴의 정의를 위한 Notation

구조스타일을 개선하기 위해 기존의 구조스타일에서 사용하던 notation의 확장이 필요하다. 여기서는 가장 보편적인 notation을 이용함으로써 처음 이 notation을 접한다 할지라도 상식에서 벗어나지 않도록 하고자 한다. 중요한 notation은 (그림 3)과 같고, 다이어그램의 표현에서 필요한 속성을 자유롭게 기술하기 위해 UML의 스테레오를 활용한다.




(그림 3) 구조 패턴의 Notation

### 3.3. 구조 패턴의 정의

구조 패턴의 이해와 활용을 높이기 위하여 다음과 같은 명세로 각각의 구조 패턴을 정의하기로 한다.

① 구조패턴의 명칭 : 구조 패턴의 이름
② 구조 패턴의 설명 : 구조 패턴의 구성적 측면과 제어적 측면의 간략한 설명
③ 응용사례 : 스타일이 적용되는 구체적인 응용사례
④ 특징 : 소프트웨어 품질관점의 특성이나 관련 패턴 등 보조적 내용을 기술

#### 3.3.1 Multi-process Invocation 구조 패턴

① 설 명

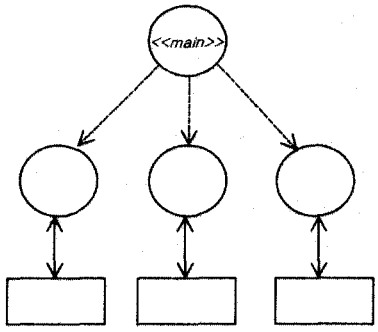
이 구조 패턴은 (그림 4)와 같은 형태로서 하나의 프로세스가 동일한 또는 데몬의 형태로 동작을 시작하게 되면 그 이후 지속적으로 동작할 복수의 프로세스를 기동시킬 때 활용되는 구조 패턴이다.

② 응용사례

UNIX에서 부팅을 할 때 'init' 프로세스가 각각의 단말기로부터 입력되는 명령어를 처리하는 'shell' 프로그램을 할당하는 경우에 사용되는 패턴이다.

③ 특 징

대부분의 복잡한 서버 프로그램에서 나타나는 유형으로, 기동된 각각의 프로세스들은 다시 새로운 구조 패턴을 형성할 수 있다.




(그림 4) Multi-process Invocation 구조 패턴

3.3.2 Delegation 구조 패턴

① 설 명

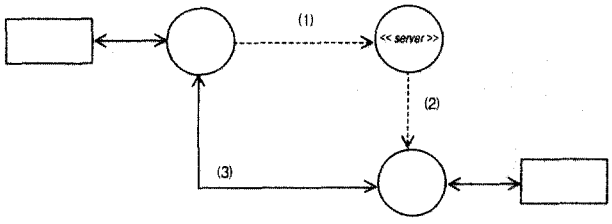
이 구조 패턴은 (그림 5)와 같은 형태로서 클라이언트-서버로 구성되는 환경에서 새로운 클라이언트의 요청이 있을 때마다 클라이언트의 요청을 서버를 대신해서 처리할 프로세스를 할당할 때 활용하는 패턴이다.

② 응용사례

클라이언트의 대부분 응용에서 전형적으로 나타난다. 예를 들어 네트워크 게임 서버가 각각의 사용자 PC로부터 입력되는 사용자 명령을 받아서 이에 대한 서비스를 전담할 프로세스를 기동시킬 때 활용된다.

③ 특 징

서버는 임의의 클라이언트가 접속할 수 있도록 well-known 주소를 가진다.




(그림 5) Delegation 구조 패턴

3.3.3 Pipe 구조 패턴

① 설 명

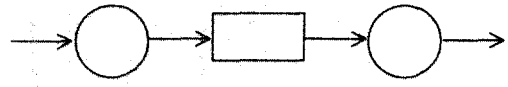
하나의 프로세스가 다른 프로세스로 데이터를 전달 할 때 사용되는 구조 패턴으로 (그림 6)과 같은 형태를 갖는다.

② 응용 사례

모든 프로세스간 통신에서 발견되는 전형적인 패턴이며, 다른 복잡한 패턴을 레고처럼 구성해 주는 컴포넌트 패턴이다.

③ 특 징

구조 스타일의 Pipes-and-filters 스타일을 구성하는 하나의 컴포넌트가 되며, 사용이 많기 때문에 이미 UNIX나 JAVA 등의 언어에서 built-in feature로 제공된다.



(그림 6) Pipe 구조 패턴

3.3.4 Reliable Data Transfer 구조 패턴

① 설 명

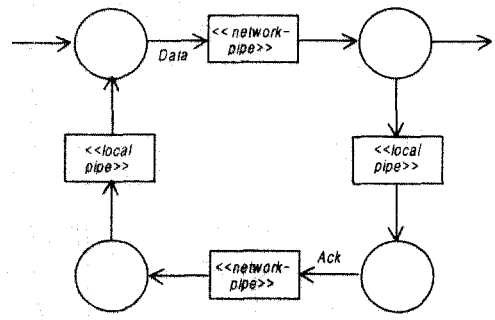
이 구조 패턴은 (그림 7)과 같은 형태로서 클라이언트-서버간 또는 Peer-to-peer간 통신에서 상대측의 수신여부를 확인해야 하는 응용에서 송신된 데이터에 대한 확인이 필요할 때 활용된다. 이를 위해 송신측과 수신측 프로세스는 통신망 채널을 이용한 정보교환을 하지만 내부적으로 데이터 처리 프로세스와 수신확인(ack) 프로세스는 벡터 배열과 같은 내부의 공유 메모리를 활용한다.

② 응용사례

서버로부터 전송되는 이미지 파일을 수신해서 사용자에게 표현하고자 하는 응용에서 발견될 수 있다.

③ 특 징

데이터 송신측 프로세스는 데이터의 입력을 두 군데서 받기 때문에 처리에서 지속적인 블록킹(blocking)이 발생해서는 안되며, 블록킹을 피하는 과정에서도 타이머의 설정시간이 성능에 영향을 줄 수 있다. 두 개의 프로세스간 양방향 통신을 위해 Bi-direction 스타일을 구성할 수도 있다.




(그림 7) Reliable Data Transfer 구조 패턴

3.3.5 Multicaster 구조 패턴

① 설 명

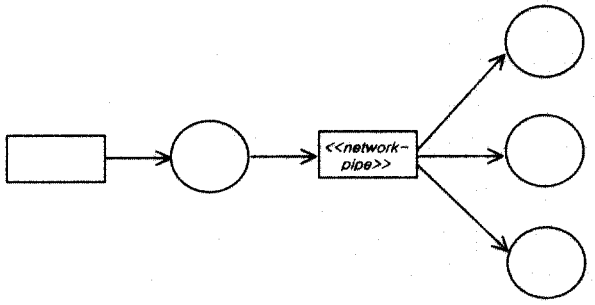
화상회의와 같이 하나의 소스로부터 입력되는 영상, 음성 정보가 회의 참여자 모두에게 동시에 전송되는 경우로 (그림 8)과 같은 형태이다. 순수한 멀티캐스트의 개념에 기초를 두어 그룹의 구성원들은 하나의 멀티캐스트 주소를 공유하며, 이 주소를 통해 한번의 데이터 전송에 의해 모든 그룹의구성원들이 데이터를 수신하게 된다.

② 응용사례

화상회의, 채팅

③ 특 징

기본적으로 실시간성 정보 전송의 경우는 전송데이터에 대해 수신확인이 불필요하지만, 신뢰성 있는 정보전송에서는 하나의 전송데이터에 대해 복수의 수신확인 메시지가 수신된다. 또한 실시간성 멀티캐스트의 경우도 정보의 흐름 제어를 위해 수신확인을 필요로 할 수 있다. 수신확인이 필요한 경우는 송신자 프로세스와 수신확인을 수신하는 수신자 사이에 내부적인 통신 채널이 필요하며, 이때는 통신망의 성능과 수신자의 수신성능을 고려하여 결과적으로 높은 안정된 정보를 제공하는 대신 송신자의 전송성능은 상대적으로 저하된다.




(그림 8) Multicaster 구조 패턴

3.3.6 Point-to-point multicaster 구조 패턴

① 설 명

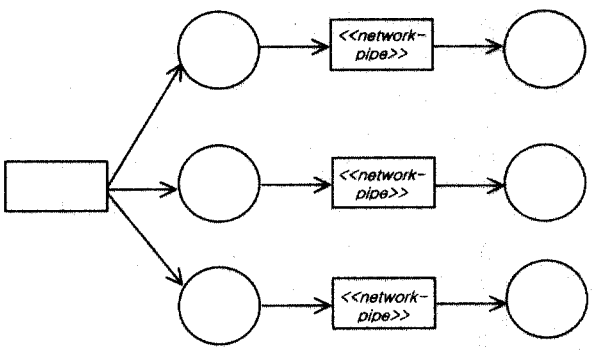
Multicast 스타일과 동일한 상황에서 사용될 수 있으나, 이 경우에는 멀티캐스트 라우터가 지원되지 않는 원거리통신망환경(WAN) 통신망의 경우에서와 같은 그룹내의 여러 수신자에게 정보를 전달하기 위해 복수의 주소가 필요한 경우로 (그림 9)와 같은 형태이다.

② 응용사례

화상회의, 채팅, 웹캐스팅

③ 특 징

멀티스레드를 활용하는 경우는 각각의 스레드가 별개의 송신버퍼를 운용할 수도 있지만 모든 스레드가 공동으로 활용하는 하나의 버퍼를 운용할 수도 있다. 두 경우 중의 활용에 대해서는 사전에 성능 측정이 필요하다.




(그림 9) Point-to-point multicaster 구조 패턴

3.3.7 Multiplexor 구조 패턴

① 설 명

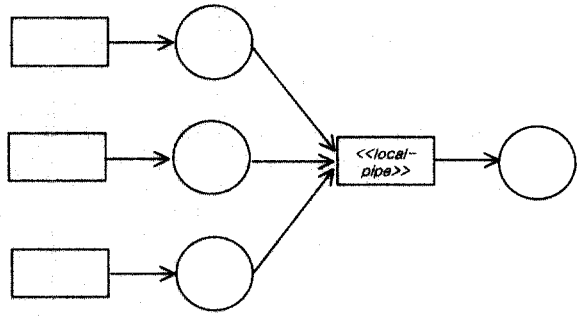
별개로 입력되는 미디어 채널의 정보를 통합해서 하나의 스트림으로 통합해서 하나의 채널정보로 만들어 전송하고자 하는 경우에서 발견할 수 있다. 예를 들어 카메라에서 입력되는 영상정보와 마이크로부터 입력되는 음성정보를 하나의 채널 안에 통합해서 전송하고자 하는 경우로 (그림 10)과 같은 형태이다.

② 응용사례

실시간 스트리밍 방송

③ 특 징

Multicast 구조 패턴과 반대 개념의 구조 패턴으로 입력 순서에 따라 출력 순서화가 이루어진다.

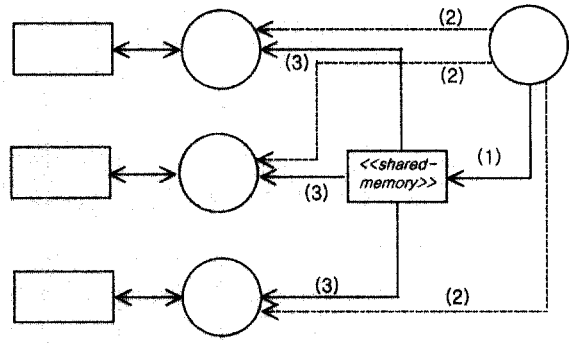



(그림 10) Multiplexor 구조 패턴

3.3.8 Event Notification 구조 패턴

① 설 명

각각의 프로세스는 외부로부터 정보의 입력을 기다리면서 블록킹 상태에 있다가 이 프로세스들을 감시하는 관리자 프로세스가 새로운 정보를 각 프로세스에게 전달해야 할 필요가 있을 사용되며, (그림 11)과 같은 형태이다.




(그림 11) Event Notification 구조 패턴

② 응용사례

각 프로세스의 감시정보(트래픽 측정주기 등)의 변경의 경우에 활용된다.

③ 특 징

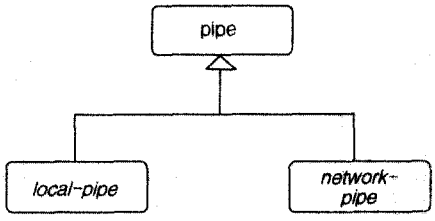
개별 프로세스를 블로킹 상태에 둘 수도 있지만 일정시간 감시하다가 입력정보가 없으면 블록킹 상태에서 벗어나 공유메모리의 정보를 자발적으로 읽도록 할 수도 있다.

### 3.4 구조 패턴간의 관련

스타일의 관련성의 발견을 구조적 컴포넌트의 조립과 분해, 확장에 도움을 준다. 대표적인 관련에는 스타일 사이의 상속관계와 연관관계가 있다.

#### 3.4.1 구조 패턴간의 상속관계

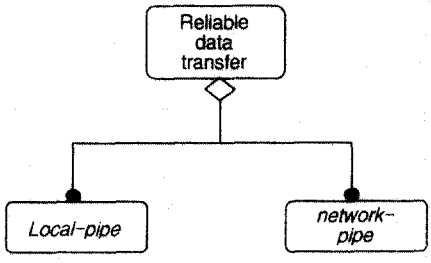
(그림 12)는 구조 패턴에서의 상속관계를 보여 주고 있다. 즉, pipe 구조 패턴은 동일한 하나의 시스템 내에서 스트림 버퍼 역할을 하는 local-pipe 구조 패턴과 통신망 상에서 두 개의 시스템간 정보전달을 하는 network-pipe 구조 패턴으로 상속되는 관계를 표현한다.




(그림 12) 구조 패턴의 상속관계

#### 3.4.2 구조 패턴간의 연관관계

(그림 13)은 구조 패턴에서의 연관관계를 보여 주고 있다. 즉, Reliable-data-transfer 구조 패턴은 local-pipe 구조 패턴과 network-pipe 구조 패턴의 결합을 통해 생성해 낼 수 있음을 알 수 있다.

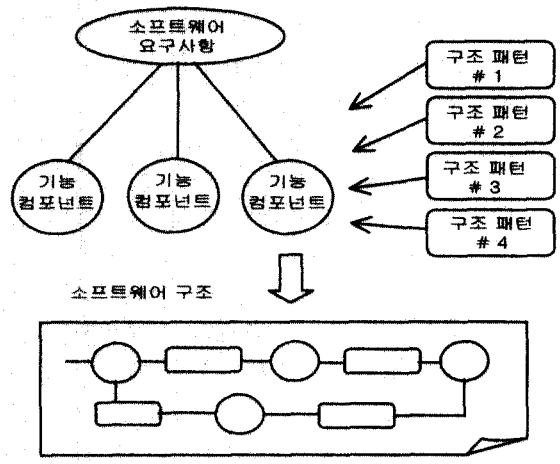



(그림 13) 구조 패턴의 연관관계

## 4.구조패턴기반 소프트웨어구조설계

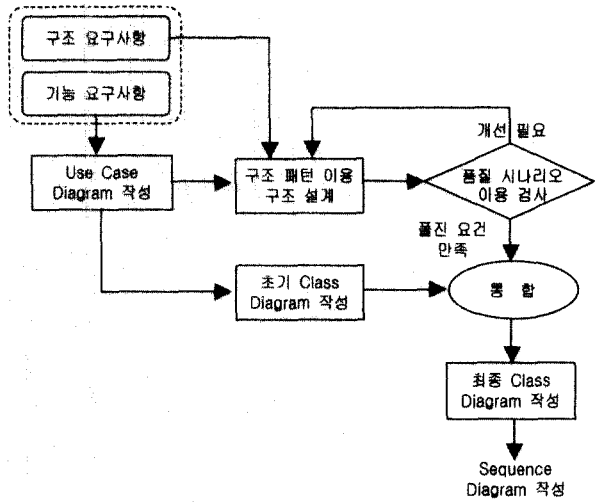
### 4.1 구조패턴을 이용한 구조설계 방법

(그림 14)는 정의된 소프트웨어 구조 패턴을 활용하여 소프트웨어 구조를 설계하는 기본개념을 보여 준다. 우선 소프트웨어의 요구사항은 기능적인 관련성이나 유사성, 기능규모, 추상화 레벨에 따라 적당한 기능 설계요소(design element)로 분해된다. 이 기능요소들은 다시 구조 패턴의 집합으로부터 각각의 기능요소에 적합한 구조 패턴을 매핑해 가면서 소프트웨어 구조가 구체적으로 설계된다.




(그림 14) 구조 패턴 이용 개념

(그림 15)는 구조 패턴을 활용한 구조설계와 UML을 활용한 상세설계를 연계시키는 과정을 설명한다. 소프트웨어의 요구사항은 크게 구조측면의 요구사항과 기능측면의 요구사항으로 구분된다. 예를 들어 채팅 시스템에서 채팅 서버를 이용하는 채팅 시스템을 구현하고자 한다면 이것은 구조적인 요구사항이 된다. 또한 채팅 참가자의 등록이나 채팅방의 개설과 같은 내용은 기능측면의 요구사항이 된다. 구조 패턴을 이용한 구조설계의 입력은 구조측면의 요구사항과 함께, 기능관점에서의 요구사항을 구체화한 Use Case 다이어그램이다. 이 두 가지 입력자료와 구조 패턴의 집합을 이용하여 소프트웨어 구조가 설계되며, 설계된 결과는 소프트웨어 품질 시나리오(Quality Scenario)를 이용하여 설계결과에 대한 확인과정을 거친다. 품질 시나리오는 '채팅 메시지가 채팅참가자에서 전달되는 방식은 합리적인가?'와 같



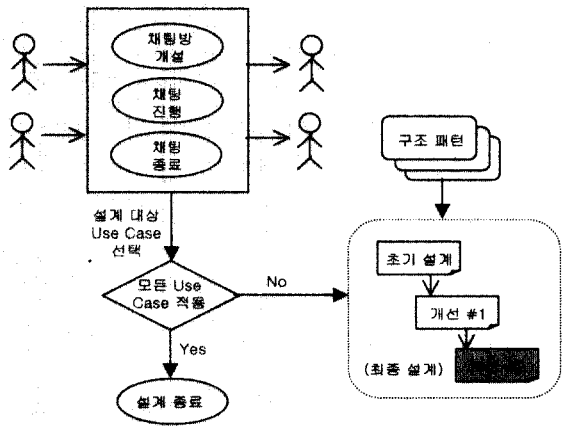

(그림 15) 구조 패턴을 이용한 설계 절차

은 질문 즉, 자극(stimulus)에 대해 적절한 응답(response)을 주는 가를 확인하고 점검하는 데 이용된다. 이 과정에서 자극에 대한 적절한 응답을 찾지 못하게 되는 경우 구조설계의 보완이 요구되며, 모든 질문이 다 통과될 때까지 설계 보완은 반복적으로 이루어진다. 최종적으로 품질 시나리오를 모두 만족되면 구조설계서도 완성된다.

한편 기능적 요구사항을 토대로 작성된 Use Case 다이어그램은 Class 다이어그램을 설계하는 데도 중요한 입력 자료로 활용된다. Use Case로부터 Class를 찾기 위해서는 서비스 시나리오를 이용하거나 또는 OMT(Object Modeling Technique)에서 제안하는 방식을 이용할 수 있으며, 이러한 방식을 통해 초기 Class 다이어그램이 작성된다. 객체지향 설계과정에서 나타나는 객체는 시나리오를 분석할 때 중요한 명사를 관찰하여 발견할 수 있는 entity 객체이외에도 사용자로부터 입력을 받거나 정보를 출력하기 하는 데 이용되는 User Interface 객체 그리고 통상 하나의 Use Case를 관리하는 Control 객체들이 있다.

물론 Use Case 다이어그램으로부터 초기 Class 다이어그램을 얻을 수는 있으나 모든 Class들을 얻었다고 보기는 어려우며, 오히려 구조설계서를 통해 또 다른 Class, 특히 제어용 클래스를 추가로 확인할 수 있다. 따라서 이 두 가지 다이어그램이 통합되어야 비로서 Class 다이어그램을 완성할 수가 있고 아울러 설계의 다음과정인 Sequence 다이어그램을 작성하는 데 필요한 모든 Class를 발견하였다고 할 수 있다.

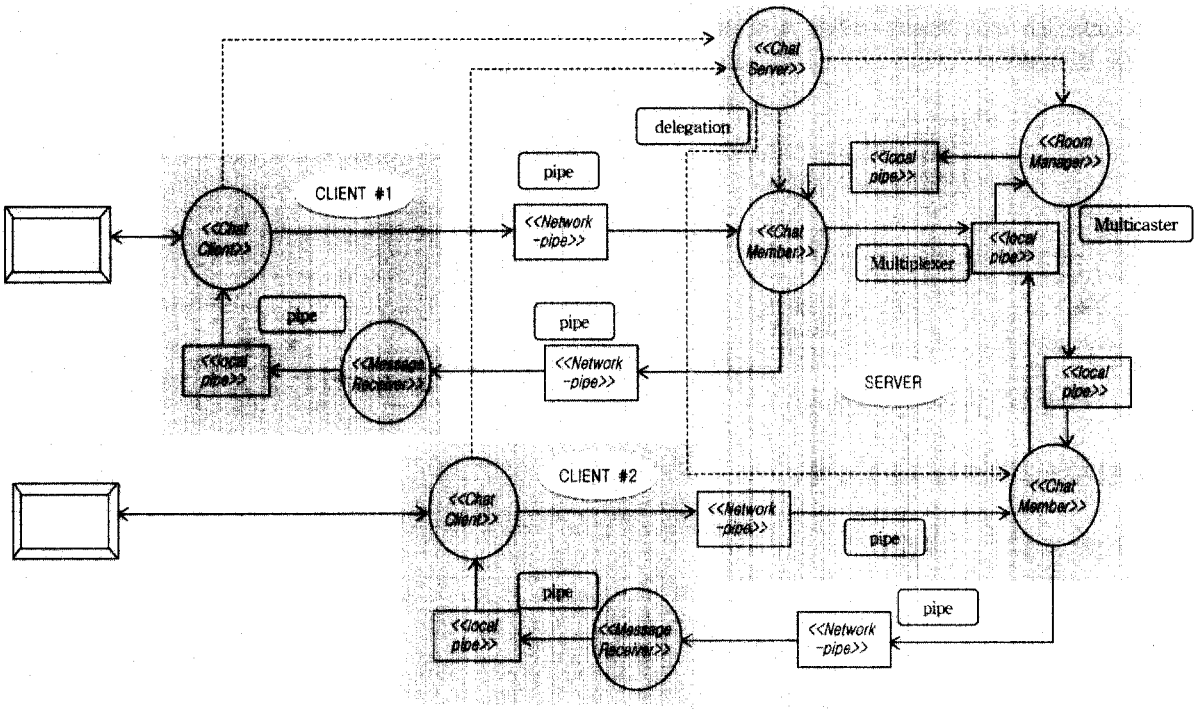
(그림 16)은 Use Case 다이어그램에 나타나는 복수의 Use Case를 어떻게 구조 패턴을 활용한 설계로 전환하여 가는가를 보여 준다. 일단 Use Case 다이어그램이 작성되면 복수의 Use Case 중 구조적인 골격을 분석하기 용이한 Use Case를 먼저 선정하여 구조 패턴의 집합을 참조해 가면서 초기의 구조를 설계한다. 이어 나머지 Use Case 중에서 구조에 영향을 줄 수 있는 사례부터 우선 순위에 따라 선택해 가면서 구조설계 내용을 개선해 간다. 만약 모든 Use Case를 다 적용하게 되면 구조설계의 과정도 종료된다.




(그림 16) Use Case를 이용한 점진적 구조 설계

### 4.2 채팅 응용에서의 구조 패턴 활용 예

(그림 17)은 하나의 채팅 서버에 2명의 사용자가 접속된 채팅 환경을 보여 주고 있다. 이 응용에서 발견할 수 있는 몇 가지 패턴을 살펴보면 다음과 같다.




(그림 17) 채팅 응용에서의 구조 패턴

(1) Delegation 구조 패턴 : 채팅 참가자로부터 서버의 Well-known 주소로 접속하는 클라이언트의 연결요청을 받으면 ChatServer 프로세스는 이 클라이언트를 담당할 ChatMember 스레드를 생성한다. 여기서 ChatServer가 ChatMember에세 서비스를 위탁하는 과정에서 우리는 Delegation 구조 패턴을 발견한다.

(2) Multiplexor 구조 패턴 : 각 채팅 참가자가 입력하는 정보는 채팅 서버로 전송되고, 이 정보는 다시 모든 참가자에게 전달된다. 여기서 복수의 ChatMember들이 하나의 RoomMenager에게 정보를 전달하는 과정에서 Multiplexor 구조 패턴을 발견한다.

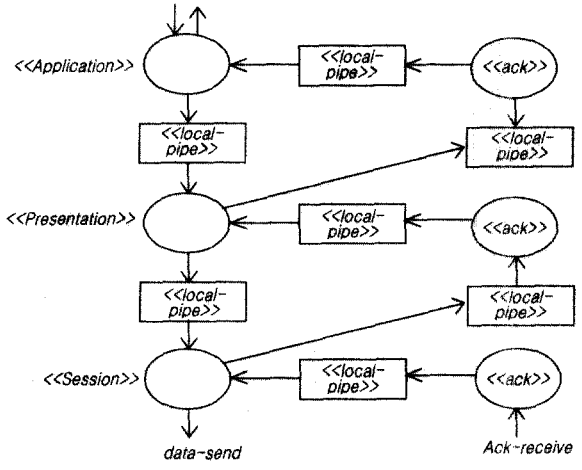
(3) Multicaster 구조 패턴 : Multiplexor 구조 패턴을 통해 입력된 정보는 다시 ChatMember들에게 분산되는데, 이 과정은 Multicaster 구조 패턴을 발견한다.

(4) Pipe 구조 패턴 : 이 응용에서 pipe 구조 패턴은 여러 부분에서 발견할 수 있다. 예를 들어, 통신망을 통한 ChatClient와 ChatMember 스레드 사이에서 채팅 메시지를 전달하는 과정이나 일단 서버에 입력된 메시지를 각 클라이언트에게 전달하는 과정 등에서 Pipe 구조 패턴이 사용되고 있음을 알 수 있다.

이 예를 통해 채팅 소프트웨어가 몇 가지 구조패턴을 활용하여 이해하기 용이한 전체적인 소프트웨어의 구조가 모습을 갖추게 될 수 있음을 알 수 있다. 이는 비단 채팅 응용 뿐 아니라 다른 많은 응용에서도 활용될 수 있다는 가능성을 보여 준다.

### 4.3 구조 패턴의 활용효과

구조 패턴은 기존의 구조 스타일과 비교하여 구체화된 패턴을 설계자에게 제공한다는 장점이 있다. (그림 18)은 Layered 구조 스타일을 Pipe 구조 패턴을 이용하여 변환 및 표현된 것이다. 이 그림은 Unix나 Java 등 프로그램 개발자들




(그림 18) Layered 구조 스타일의 구조 패턴 이용 표현

이 친숙한 Pipe를 이용함으로써 이 구조도를 참고하여 바로 프로그래밍이 가능하다는 것을 알 수 있다.또한 구조 패턴에서는 소프트웨어의 실행환경이 실제 동시적 관점이 중요하게 고려되기 때문에 이러한 동시적 관점에서 소프트웨어 구조에 대한 이해가 매우 용이하다.

구조 패턴을 활용하는 또 다른 효과는 디자인 패턴을 활용하고자 할 경우 전환이 매우 용이할 정도의 구체화된 표현이 제공된다. 이외에도 앞에서 언급하였듯이 UML을 이용한 설계와 연계가 용이하며, 아울러 제어기능의 객체를 발견하는 데도 효과적인 점들이 있다.

## 5. 결론 및 향후 연구과제

이 논문은 구조 패턴에 대한 개발배경이나 개념, 기본적인 패턴들을 소개하고 있다. 이러한 구조 패턴은 소프트웨어 구조를 설계하기 위한 컴포넌트로 유용할 것으로 기대된다. 그러나 아직도 많은 응용 안에 감추어 진 구조 패턴들이 더 있을 것으로 보아, 이에 대한 추가적인 발견과 명세가 필요하다. 또한 구조설계언어를 이용한 구조 패턴의 명세도 추진되어야 한다. 구조 패턴들의 품질은 연구과제로 남겨져 있다. 소프트웨어 구조설계에서 대안으로 제시될 수 있는 구조 패턴들 사이에서 어떤 상황에 어떤 패턴이 더 적합한지가 품질 속성에 따라 시험되고 정의되어야 한다. 또한 이제까지의 구조 패턴의 연구가 프로세스나 스레드의 관점이었다면 데이터에 대한 좀 더 면밀한 분석과 도입이 추진되어야 한다.

특히 구조 패턴을 프로그래밍으로 연결시키기 위한 부분도 필요하게 되는 데, 방법으로는 pipe와 같이 built-in feature로 만들 수도 있고 경우에 따라서는 디자인 패턴과 같이 예문을 제시할 수도 있다. 결국 이러한 구조 패턴은 경험이 많지 않은 소프트웨어 개발자에게 구조를 설계하는 데 많은 도움이 될 것으로 생각된다. 그리고 많은 경험을 갖춘 경험자라 하더라도 이러한 구조 패턴들의 정리된 목록을 활용하여 보다 편리하게 소프트웨어 설계를 추진할 수 있을 것이라고 본다.

## 참 고 문 헌

[1] Acme team, Overview of acme project, http://www-2.cs.cmu.edu/~acme, CMU.

[2] Erich Gamma, Richard Helm, Ralph Johnson and John Vissides, Deisgn Patterns, Addison Wesley, Jan., 2000.

[3] Felix Bachmann, Len Bass, Gay Chastek, Patric Donohoe, Fabio Perzzi, Architecture Based Design Method-CMU/SEI-2000-TR-001, Jan., 2000.

[4] Frank Buschmann, Regine Meunier, Hans Rohnert, Perter Sommerlad, Michael Stal, Pattern-Oriented Software Ar-

chitecture Volume 1 : A System of Patterns, John Wiley & Sons, July, 2001.
[5] George T. Heineman & William T. Councill, Component-Based Software Engineering-Putting The Pieces Together, Addison-Wesley, 2001.
[6] Jonathan Adams, Srinivas Koushik, Guru Vasudeva, George Galambos, Patterns for e-business, IBM Press, Oct., 2001.
[7] Jon Siegel and OMG Staff Strategy Group, Developing in OMG's Model-Driven Architecture, Object Management Group White Paper, November, 2001.
[8] Kert C. Wallnau, Scott A. Hissam, Robert C. Seacord, Building Systems from Commercial Components, Addison-Wesley, 2002.
[9] Len Bass, Paul Clements, and Rick Kazman, Software Architecture in Practice, 1998.
[10] Mary Shaw and David Galan, Software Architecture-Perspective on an Emerging Discipline, Prentice Hall, 1996.
[11] Open Group, The Open Group Architectural Framework (TOGAF)-Version 7, 2001.
[12] Robert T. Monroe, Andrew Kompanek, Ralph Melton, and David Galan, Architectural Styles, Design Patterns, and Objects, IEEE Software, pp.43-52, January, 1997.

**궁 상 환**

e-mail : kung@cheonan.ac.kr
1977년 숭실대학교 전자계산학과(이학사)
1983년 고려대학교 전자정보처리학과 (경영학석사)
1998년 충북대학교 전자계산학과 졸업 (이학박사)
1977년~1981년 제2군수지원사령부 제원처리실 프로그램장교
1981년~1998년 한국전자통신연구원 책임연구원
1996년~1997년 미국 스탠포드 연구소(SRI) 초빙연구원
1998년~2000년 중소기업청 정보화지원과 전산사무관
2000년~2000년 미국 카네기 멜론 대학 SE 과정 수료
2000년~현재 천안대학교 정보통신학부 조교수
관심분야 : 소프트웨어 구조, 분산시스템