한국산학기술학회논문지
Vol. 8, No. 1, pp. 82-95, 2007

품질속성을 고려한 소프트웨어 아키텍처 패턴의 정의

궁상환[1*]

Definition of Architecture Patterns regarding Quality Attributes

Sang-Hwan Kung[1*]

요 약 본 논문은 소프트웨어의 아키텍처 설계에서 많이 활용될 수 있는 아키텍처 패턴들을 분류하고 명세화하는 방법을 주제로 한다. 이를 위해 먼저 아키텍처 패턴의 활용과 관련한 개선된 아키텍처 개발방법론을 제시하고, 이러한 방법론을 지원하기 위한 패턴의 분류와 정의방법을 제안한다. 패턴은 추상화 레벨이나 응용 도메인에 따라 매우 다양하기 때문에, 여러 가지 각도에서의 분류체계를 제시하여 향후 분류 및 저장된 아키텍처 패턴을 쉽게 검색하고 활용할 수 있도록 고려하였다. 또한 패턴의 표현이 다양하여 패턴의 이해가 용이하지 못한 점을 고려하여 패턴의 추상화 개념을 도입하고, 패턴간의 관계를 정의하는 방법을 제시하여 패턴들의 핵심적인 구조와 역할에 대한 이해를 돕고자 하였다. 아울러, 아키텍처 패턴을 선정할 때의 중요한 고려사항이 되는 품질속성도 기존의 범용적인 개념에서 아키텍처 지향적인 개념으로 확장이 가능하도록 하여 각각의 아키텍처 패턴을 정의할 때 보다 명확한 아키텍처의 품질에 대한 명세가 함께 설명되도록 하였다.

Abstract The paper focuses on how to classify as well as to define the Architecture Patterns which are popularly used in the design of software architecture. In order to achieve this purpose, we propose not only the revised methodology for Pattern-Oriented Software Architecture Design, but also new method of classification and definition for the Architecture Patterns. Especially, because the patterns are so diverse depending on the level of abstraction and types of applications, it was considered to have some different views of classification of the patterns in order to support convenient access to classified and stored patterns. The abstraction of the pattern is another important result of the research, which is devised for concrete expression of the patterns and for presentation of the interrelation among group of the patterns. The research also includes the extension of the quality model popularly adopted in the software domain, which enables the description of the patterns with the well defined quality attributes in terms of software architecture's point of view.

Key Words : 소프트웨어 아키텍처, 아키텍처 스타일, 아키텍처 패턴, 품질속성, 아키텍처 평가, 디자인 패턴

1. 서론

본 연구는 소프트웨어의 아키텍처 설계과정에서 아키텍처 접근법을 정의하는 데 유용하게 활용될 수 있는 아키텍처 패턴들을 분류하고 정의하는 것을 다룬다. 종종 아키텍처 패턴 기반의 설계는 컴포넌트 기반 설계와 비교되기도 하는 데, 패턴은 내용물이 채워지지 않은 구조요소 간의 관계에 대한 개념이라면, 컴포넌트는 내용물이 채워진 요소에 대한 개념이라고 보면 될 것이다. 컴포넌트는 기본적으로 수정을 전제로 하지 않는 반면, 패턴의 구현에서 구성요소는 다양한 솔루션으로 구현될 수가 있는 것이다[7][11].

아키텍처 패턴은 그 종류가 많고 또한 패턴의 정의방법에서 일관성이 없어 이 패턴을 기억해서 활용하는 것이 용이하지 않다. 본 연구는 그동안 다양한 관점과 레벨에서 정의되고 있는 패턴들을 분류방법 및 정의내용에 있어서 체계화하기 위한 시도이다. 이러한 패턴의 체계화는 사용자가 패턴의 검색 및 활용이 용이하게 될 뿐 아니라 각각의 패턴의 특성을 기억하기가 용이하여 패턴을 보다 더 쉽게 사용하는 것이 가능하게 된다. 또한, 연구의 또 다른 성과라면 패턴들의 관계를 분석하여 모든 패턴이 별개적인 것이 아니라 상호 연관관계가 있으며, 이러한 관계를 이해하게 되면 모든 패턴을 암기하는 방식으로 기억하는 것이 아니라 개념의 이해를 통해 보다 간편

이 논문은 2004년도 한국학술진흥재단의 지원에 의하여 연구되었음(KRF-2004-003-D00347)
[1]백석대학교 정보통신학부
*교신저자: 궁상환(kung@bu.ac.kr)

하게 활용할 수가 있다는 장점이 있다고 하겠다. 그러면 어떻게 패턴을 쉽게 이해하고 기억할 수 있을 까? 본 연구에서는 다양한 패턴을 쉽게 이해하도록 하기 위해서는 이러한 패턴들의 중요한 속성을 토대로 체계적으로 패턴을 분류하고, 각각의 패턴의 특징을 여러 가지 관점에서 정확히 기술하는 것이 필요하다 하겠다.

연구에서는 먼저 아키텍처 패턴의 활용을 제고하기 위한 아키텍처 패턴 기반의 설계방법을 제안하였다. 이 과정에서 아키텍처 패턴이 어떠한 방법으로 활용되며, 어떠한 요소들이 고려되어야 하는 가를 파악할 수 있도록 하였다. 또한 아키텍처 패턴의 분석 및 정리를 위해 객체지향 프로그램을 위해 최근 많이 보급되고 있는 소프트웨어 디자인 패턴과 참고문헌에서 아키텍처 패턴에 대한 예제들을 분석하였다[4][6]. 아울러 메시징 엔진, 워크플로우 엔진 등의 미들웨어 소프트웨어의 아키텍처 분석을 통해 아키텍처 스타일과 아키텍처 패턴들의 발굴 및 분류를 위해 노력하였다[1][2][14].

아키텍처 패턴의 분류와 정의는 현재까지 수행된 많은 패턴 기반 연구에서 발견한 패턴 뿐 아니라, 앞으로 더 연구될 많은 패턴들을 쉽게 접근하고 활용하게 할 수 있는 기반이 된다. 연구에서는 아키텍처 패턴의 분류를 위해 각각의 패턴들을 유형별로 그룹화 하는 방안을 검토하고, 특징이나 특성이 상호 연관된 패턴들을 해당 그룹으로 분류하였다.

패턴의 정의를 위해서는 다른 패턴의 명세방법을 비교 및 검토하고, 또한 패턴 기반의 아키텍처 설계방법에서 요구하는 내용들이 사전에 패턴의 정의에서 고려될 수 있도록 명세항목을 설정하였다. 특히 이 명세에서는 아키텍처 패턴의 품질속성이나 아키텍처 결정사항을 체계적으로 포함시켜, 아키텍처 설계에서 아키텍처 접근법으로의 전환이 용이하도록 하였다. 또한 명세에서는 패턴의 구조를 UML 기반으로 다이어그램화 하기 위한 표현방법에 대해서도 검토하여 보았다.

아키텍처 패턴들의 관계는 패턴의 기본속성을 이해하여 조합, 확장 시키는 데 유용한 개념이 될 수 있다. 따라서 연구에서는 아키텍처 패턴들의 상호 관련성을 쉽게 분석하고 표현할 수 있도록 패턴간의 연관, 상속 관계 등을 정의하여 패턴의 지속적인 연구와 발전에 활용될 수 있도록 하였다.

2. 아키텍처 패턴 관련연구

2.1 아키텍처 설계방법론

소프트웨어 아키텍처의 설계는 소프트웨어 시스템의 개발에서 매우 중요한 작업이다. 특히 아키텍처는 대규모의 소프트웨어 뿐 아니라, 하나의 기본적인 아키텍처에서 다양한 기능을 갖춘 여러 가지 시스템이 만들어 질 수 있는 제품라인을 개발하는 경우에는 더욱 중요하다. 그럼에도 불구하고, 아키텍처를 정의하고 관리하는 방법이 명확하게 정의되지 않은 가운데 소프트웨어가 개발되는 경우가 많은 것도 사실이다. 이러한 이유로 인해, 아키텍처의 설계를 위한 명확한 절차를 정립하고, 이 절차에 따라 체계적인 아키텍처를 설계하는 것이 필요하다.

대표적인 아키텍처 설계방법에는 아키텍처 기반 개발방법(Architecture-Based Development, ABD)과 아키텍처 기반 설계방법(Architecture-Based Design Method, ABDM), 그리고 품질속성 기반 설계방법(Attribute- Driven Design, ADD)이 있다.

ABD는 대규모 소프트웨어나 제품라인(Product Line)을 중심으로 한 아키텍처 개발방법이다. 이 방법은 아키텍처의 설계 뿐 아니라 이후의 아키텍처의 구현 및 유지보수에 이르기까지 전체의 소프트웨어 개발공정을 포괄적으로 설명한다. 즉, 아키텍처를 기반으로 한 소프트웨어 생명주기를 다루는 방법이라고 할 수 있다. ABD는 아키텍처의 요구사항 분석, 아키텍처 설계, 아키텍처 문서화, 아키텍처 분석, 아키텍처 구현, 아키텍처 유지보수의 여섯 단계로 구성된다[12].

ABDM은 비즈니스 컨텍스트와 품질, 그리고 기능적 요구사항을 입력으로 개념적인 아키텍처 설계를 수행하는 방법이다. ABDM은 기본적으로 세 가지 기반을 가지고 실행된다. 첫째는 기능의 분해이다. 이는 커플링(Coupling)과 코히젼(Cohesion) 기반의 기법이다. 두 번째는 아키텍처 스타일의 선택을 통하여 품질과 비즈니스 목표를 실현하는 것이다. 세 번째는 소프트웨어 템플릿의 사용이다. 소프트웨어 템플릿은 시스템을 구성하는데 사용된다. 소프트웨어 템플릿은 특정한 형태의 소프트웨어 요소가 될 수 있는 것을 정의하며, 예를 들어 아키텍처 패턴도 이 템플릿에 포함된다[5].

이상의 두 가지 방법과 여러 가지 면에서 유사한 특징을 갖는 ADD는 크게 세 가지 과정을 통해 수행된다. 이 과정에는 분해할 모듈을 선택하는 과정과 선택한 모듈을 분해 및 정제하는 과정, 그리고 마지막 과정으로 더 이상 분해가 불가능할 때까지 다시 모듈을 분해하는 과정이 포함된다. 이 중 핵심과정인 두 번째 과정은 다시 다섯 단계로 수행된다. 먼저 해당 모듈의 아키텍처 드라이버를 분석하고, 이 드라이버를 충족시킬 수 있는 아키텍처 스타일을 선택한다. 이후, 이 스타일에 기능을 할당한 뒤, 하위 모듈의 인터페이스를 결정한다. 마지막 단계는 쓰임새(Use Case)와 품질속성을 정제하고, 검증하여 하위 모

들의 제약사항을 만드는 과정이 수행된다[17].

2.2 아키텍처 품질

아키텍처의 품질은 최종적으로 개발될 시스템이 만족시켜야 할 중요한 요구사항이다. 개발 초기에 아키텍처에 부과된 요구 품질은 아키텍처 설계과정에서 모듈이 세분화되면서 점차 더욱 구체적이고 정밀한 품질 요구사항으로 변환된다. 특히 아키텍처 설계를 위한 품질은 각각의 품질요소마다 상호 배타적일 수 있어, 하나의 품질을 만족하려면 다른 품질을 희생해야 하는 경우가 많다. 따라서 아키텍처 설계에서는 특정한 품질 하나의 민감한 경우도 반영해야 하겠으나, 복수의 품질 사이의 절충관계로 고려해야만 한다.

소프트웨어 시스템에 대한 일반적인 품질의 정의는 *ISO/IEC-9126*이나 *IEEE Std 1061-1998*, *McCall* 모델 등에서 제시되고 있다. 이 모델은 소프트웨어의 설계나 구현물뿐만 아니라, 소프트웨어 아키텍처의 품질을 위해서도 동일한 척도로 사용되고 있다. ISO/IEC-9126 모델에서는 품질을 내부와 외부로 분리한다. 소프트웨어 개발과정 중에 나오는 산출물들을 측정하기 위해서는 내부 메트릭(metric)을 사용하고, 개발이 완료된 소프트웨어 제품의 특성을 측정하기 위해서는 외부 메트릭을 사용한다. 이 모델은 품질을 6개의 품질특성으로 구분하고, 각 특성들을 다시 품질 부특성(sub characteristics)로 세분화한다. 표 1은 이 모델의 품질특성을 나열하고 있다. 한편, IEEE Std 1061-1998 표준을 기반으로 한 품질모델은 4개의 계층모델 즉, 품질(Quality), 품질특성(Quality Factor), 품질 부특성(Quality SubFactor), 그리고 계량척도(Metric)로 구성된다. 최상위 레벨의 품질이란 목표시스템이 궁극적으로 만족해야 할 품질을 의미한다. 두 번째 레벨의 품질 특성은 시스템의 외부로 나타나는 특성을 의미한다. 또한, 세 번째 레벨의 품질 부특성은 시스템 내부에서 다루는 품질로, 품질요소를 측정 가능한 소프트웨어 속성으로 변환한 것을 말한다. 끝으로, 마지막 레벨의 계량척도는 평가자 관점의 품질이며, 품질을 측정하는 방법과 척도를 의미한다.

표 1. 소프트웨어의 품질특성

품질 특성	품질 부특성
기능성	완전성, 정확성, 보안성, 호환성, 상호운영성
신뢰성	무결성, 가용성, 오류허용성
사용용이성	이해성, 학습성, 조작성, 대화성
효율성	시간경제성, 자원경제성
유지보수성	수정용이성, 확장성, 시험용이성
이식성	하드웨어 독립성, 소프트웨어 독립성, 설치용이성, 재사용성

2.3 아키텍처 스타일과 패턴

스타일 또는 패턴이란 어떤 분야에서 계속 반복해서 나타나는 문제들을 해결해 온 전문가들의 경험을 모아서 정리한 것이다. 따라서 패턴을 활용하면 전문가들이 검증해 놓은 경험을 새롭게 당면한 유사한 문제에 적용하여 문제를 쉽게 해결할 수가 있다. 스타일이나 패턴은 문학이나 음악, 건축 등 다양한 분야에서 나타나며, 소프트웨어의 아키텍처에서도 발견된다. 앞서 설명한 아키텍처의 설계과정을 보면, 설계대상의 기능을 위한 아키텍처 스타일을 선택하고, 기능을 분해 시 선택된 스타일에 매핑하는 과정이 있다[5][12][13]. 이것은 소프트웨어 시스템이 구조적인 측면에서 특정한 스타일을 가지며, 이 스타일은 새로운 소프트웨어의 설계 시 매우 유용하게 활용된다는 것을 의미한다.

소프트웨어 아키텍처에서 패턴은 스타일과 혼용해서 사용하는 용어로, 소프트웨어 구성요소(예, 프로세스)와 요소 간 상호작용 패턴(데이터 및 신호전달)을 표현한다. 스타일이나 패턴과 관련된 용어로 참조모델이 있다. 참조모델은 표준화된 기능의 분해모델이며, 잘 알려진 분야를 표현하는 데 활용된다. 참조 아키텍처는 이 참조모델과 아키텍처 패턴으로부터 파생되며, 참조모델에 구성요소가 매핑 된 것을 말한다. 이와 관련하여, 특정한 분야에서 활용되는 소프트웨어는 유사한 아키텍처 스타일을 사용하게 되는 데, 이 경우 참조가 되는 아키텍처는 도메인 아키텍처라고 부른다[9].

2.3.1 아키텍처 스타일/패턴의 유형

아키텍처 스타일/패턴은 소프트웨어를 구성하는 요소 간의 구조적인 패턴에 따라 시스템의 그룹을 분류하는 것이다. 더욱 구체적으로 설명하면, 컴포넌트와 컨넥터의 유형에 대한 용어와, 그리고 이들이 어떻게 결합되는 가에 대한 제약조건의 집합을 정의한다.

1) 디자인 패턴

디자인 패턴은 객체지향 소프트웨어의 구조 설계 및 세부 설계를 지원하여 유연성이 있고 재사용이 가능한 객체지향 소프트웨어를 가능하게 한다. 즉, 객체지향의 특성인 클래스 또는 객체의 개념이나 상속, 연관 등 클래스간의 관계성, 인터페이스의 이용 등을 통해 개개의 객체를 모듈화 하는 차원에서 여러 클래스들의 조립을 통해 경우마다 hard coding 하는 대신 단순히 파라미터 등을 활용해서 일련의 클래스의 집합을 재사용 할 수 있도록 하여 soft coding이 가능하도록 도와준다[4]. 최근의 디자인 패턴에 대한 연구는 스레드를 기반으로 하는 병

렬처리 환경에서의 디자인 패턴을 포함한다[18]. 이것은 사실상 아키텍처 스타일이나 패턴에 포함될 내용이나, 프로그래밍까지 연결되는 상세한 표현을 하고 있고, 특히 스레드 사이에 함수 호출이나 자료 접근이 가능하다는 점으로 인해 디자인 패턴의 범위에 포함시켜야 하는 지 혹은 아키텍처 패턴의 범위에 포함시켜야 하는 지 판단하기 어려운 부분이다. 여하튼 스레드 기반의 아키텍처 패턴은 소프트웨어 설계자나 프로그래머가 그 아키텍처를 바로 프로그램에서 이용할 수 있다는 커다란 장점을 제공한다는 점이 중요하다.

2) 아키텍처 스타일

아키텍처 스타일은 소프트웨어 아키텍처에서 전형적으로 많이 나타나는 패턴의 예를 소개하고 있다. 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 모듈 간에 상, 하위의 계층구조를 형성하는 아키텍처 스타일이다[18].

3) 아키텍처 패턴

아키텍처 스타일이 발표된 이후, 많은 연구를 통해 더욱 구체화되고 세분화된 패턴들이 발표되었다. 이러한 패턴들에는 임베디드 시스템을 위한 실시간 아키텍처 패턴이나 비즈니스 응용을 위한 엔터프라이즈 아키텍처 패턴, 분산환경의 응용을 위한 아키텍처 패턴, 웹서비스를 위한 웹서비스 패턴, 그리고 EJB(Enterprise Java Beans)나 특정한 프로그램 언어와 관련된 패턴까지 다양하게 개발되어 오고 있다.

2.3.2 아키텍처 패턴의 정의

1) 패턴의 정의항목

아키텍처 패턴의 유형은 소프트웨어 설계대상의 추상화 레벨이나 패턴의 용도 등에 따라 매우 다양하다. 따라서 이러한 패턴을 정의하는 항목은 관련 패턴을 기술하는 문헌별로 차이가 있다[4][6]. 패턴을 정의하는 참고자료의 조사를 통해 패턴의 정의항목을 요약한 내용은 표 2에 정리되어 있다.

2) 패턴의 구조 및 행위 표현방법

아키텍처의 구조와 행위에 대한 명세화는 구조 자체의 검증과 아울러 소프트웨어 개발의 자동화를 가능하게 한다는 점에서 점차 그 중요성을 더해 가고 있다[1][10]. 아키텍처 설계를 위해 사용되어 온 아키텍처 다이어그래밍 방법은 소프트웨어의 특성이나 또는 개발조직 등에 따라 매우 다양하게 사용하여 왔다. 대체적으로 비정형적인 다이어그램이 많이 활용되어 왔으나, 최근에는 소프트웨어 설계에 공통적으로 많이 활용하는 UML 다이어그램이 아키텍처 패턴에도 점차 보편화 되어가고 있다.

아키텍처 패턴의 표현은 무엇보다 구조적 관점을 나타내는 것이 필요하다. 이를 위해서는 구조적 다이어그램의 가장 기본적인 요소인 소프트웨어 구성요소를 나타내는 것이 필요하다. UML은 아키텍처 구성요소를 상황에 맞도록 제공할 수 있도록 적절한 메뉴를 제공한다. 패키지 다이어그램은 개념적인 소프트웨어 모듈을 표현하는 데 적합하다. 특히 분석 및 설계단계에서 나타내어야 하는 서브시스템이나 블록 등을 나타낼 때 패키지 다이어그램은 유용한 도구가 된다. 구현된 모듈의 단위 즉, 컴포넌트를 표현하는 데는 컴포넌트 다이어그램이 매우 적합하다.

특히 클래스 다이어그램은 아키텍처와 아키텍처 패턴을 명세화하는 데 매우 유용한 도구가 된다고 할 수 있

표 2. 패턴의 정의항목

디자인 패턴	아키텍처 패턴 명세사례 1	아키텍처 패턴 명세사례 2	아키텍처 접근법
o 패턴의 목적 o 별명 o 동기 o 응용 o 구조 o 구성요소 o 협력 o 결과 o 구현 o 참조코드 o 활용사례 o 관련 패턴	o 패턴의 개요 o 적용문제 o 패턴의 구조 o 협력 및 역할 o 활용효과 o 구현전략 o 관련패턴 o 샘플모델	o 패턴의 정의 o 사례 o 문맥 o 문제 o 해결책 o 구조 o 동적 시나리오 o 구현 o 변형 o 활용사례 o 결과	o 시나리오 o 속성 o 환경 o 자극 o 반응 o 아키텍처 결정 - 민감점 - 절충점 - 위험요소 - 비위험요소 o 추론 o 다이어그램

다. 클래스 다이어그램에서의 클래스는 하나의 클래스나 객체가 될 수도 있으며, 때에 따라서는 클래스나 객체들의 집합과 같이 더 큰 모듈이 될 수도 있다. 더욱이 굵은 선으로 표시될 때는 프로세스나 쓰레드와 같은 동적 프로그램 모듈을 표현할 수 있기 때문에 아키텍처의 동시성 뷰를 표현하는 데 매우 적합한 다이어그램이 된다고 할 수 있다.

그러나 아키텍처 패턴은 구조적 관점의 패턴 외에도 행위 중심적인 패턴도 있다. 또한 구조적 관점의 패턴도 구현으로 연계되기 위해서는 동적 관점의 처리 즉, 각 구성요소의 행위나, 구성요소 간 사건을 묘사하는 것도 필요하다. UML에는 이러한 동적 관점을 표현하기 위한 도구로 협력 다이어그램과 순차 다이어그램이 제공된다.

협력 다이어그램은 시간적인 개념이 표현되지 않지만 구성요소 간의 상호작용을 한 눈에 파악하게 한다는 장점을 제공하며, 이에 비해 순차 다이어그램은 시간 축 상에서 사건이 어떻게 진행되는 가를 보여 준다. 아키텍처 구성요소의 알고리즘을 표현하기 위해서는 활동 다이어그램과 같은 다이어그램도 유용한 다이어그램으로 볼 수 있다.

3. 패턴지향 아키텍처 설계방법

관련 연구에서 소개한 아키텍처 설계를 위한 3가지 방법론은 나름대로의 장단점을 가지고 있다. ABD는 아키텍처 설계결과에 대한 평가과정이 분석절차에 포함되어 있어 보다 견고한 설계가 가능한 반면, 구체적이고 세부적인 방법과 절차에 대한 설명은 부족하다. ABDM과 ADD는 아키텍처 뷰나 패턴을 설계과정에서 반영하면서 세부적인 절차를 명시하고 있으나 아키텍처 평가에 대한 설명은 부족하다는 단점이 있다. 이러한 각 방법론의 특징을 분석해 볼 때, 이들의 장단점을 반영한 새로운 방법론의 제안이 필요하다고 생각할 수 있다.

아키텍처 설계는 시스템이 만족해야 하는 기능요구사항과 비 기능 품질 속성요구사항을 만족시키는 아키텍처 접근법의 식별을 통해 점진적으로 진행된다. 본 연구에서 제안하고자 하는 패턴지향 아키텍처 설계방법 (Pattern-Oriented Software Architecture Design, POSAD)은 두 가지 관점에서 특징을 찾아 볼 수가 있다. 한 가지는 패턴지향이라는 용어 속에서 찾아 볼 수가 있는 데, 이것은 요구사항을 만족하는 아키텍처의 부분을 설계할 때 정리된 아키텍처 패턴의 정보를 이용한다는 것을 의미한다. 다른 한 가지는 설계과정 중에 평가과정을 도입하여 아키텍처의 전체 모습이 완성되기 이전에 핵심적인 아키텍처 접근법들을 검증을 하면서 설계를 지속해 나간다는 것이다. 기존의 중요한 아키텍처 평가방법들이 아키텍처 설계 후의 평가를 강조하는 것과는 차이가 평가 시점에서 차이를 갖고 있다[8][18].

이러한 방법론들을 토대로 하여 본 연구에서 제안하는 패턴기반 아키텍처 설계 절차는 그림 1과 같다.

(1) 설계가 착수되면 먼저 시스템의 비즈니스 컨텍스트를 작성한다.
  - 비즈니스 컨텍스트에는 시스템에 출현하는 행위자나 중요한 구성요소가 표현된다.
  - 구체적인 비즈니스 절차의 이해를 돕기 위하여 일반적인 시나리오나 또는 실제 상황을 근거로 한 예제 시나리오가 작성된다.

(2) 아키텍처 요구사항을 작성한다. 기능 요구사항과 비기능 요구사항을 작성한다.
  - 요구사항은 20 개 정도로 핵심적인 요구사항들을 포함하도록 한다.

(3) 비기능 요구사항을 토대로 품질속성 유틸리티 트리를 작성한다.

(4) 기능 요구사항을 토대로 설계하고자 하는 아키텍처 뷰를 선택한다.

(5) 해당 기능을 선택한 뷰에 수록하기 위해 참고되어야 할 패턴을 검색한다.
  - 검색된 패턴을 중심으로 품질요소를 중심으로 한 패턴의 평가와 적합성을 검증하는 가정을 거친다.

(6) 설계 뷰와 품질속성 유틸리티 트리, 그리고 발견한 아키텍처 패턴을 이용하여 아키텍처 접근법 분석서를 작성한다.

(7) 아키텍처 접근법 분석 결과를 토대로 아키텍처를 평가한다.
  - 평가결과가 품질요건을 만족하지 못하는 경우 아키텍처 접근법 분석절차를 다시 수행하게 된다.

(8) 평가과정을 통과하게 되면, 해당 뷰를 설계한다.
  - 즉, 분해 기능을 아키텍처 접근법에 매핑시킨다.

(9) 뷰를 동기화 한다.
  - 이번에 선택되지 않은 뷰들도 설계의 깊이가 일치하도록 설계를 구체화 한다.

(10) 설계가 완료되었는가를 검토한다.
  - 설계가 완료되었다는 의미는 시스템이 적절한 단계에서 분해가 완료되었으며, 충분히 균형있게 아키텍처가 설계되었다는 것을 의미한다.

(11) 설계내용을 문서로 작성하고 설계작업을 종료한다.

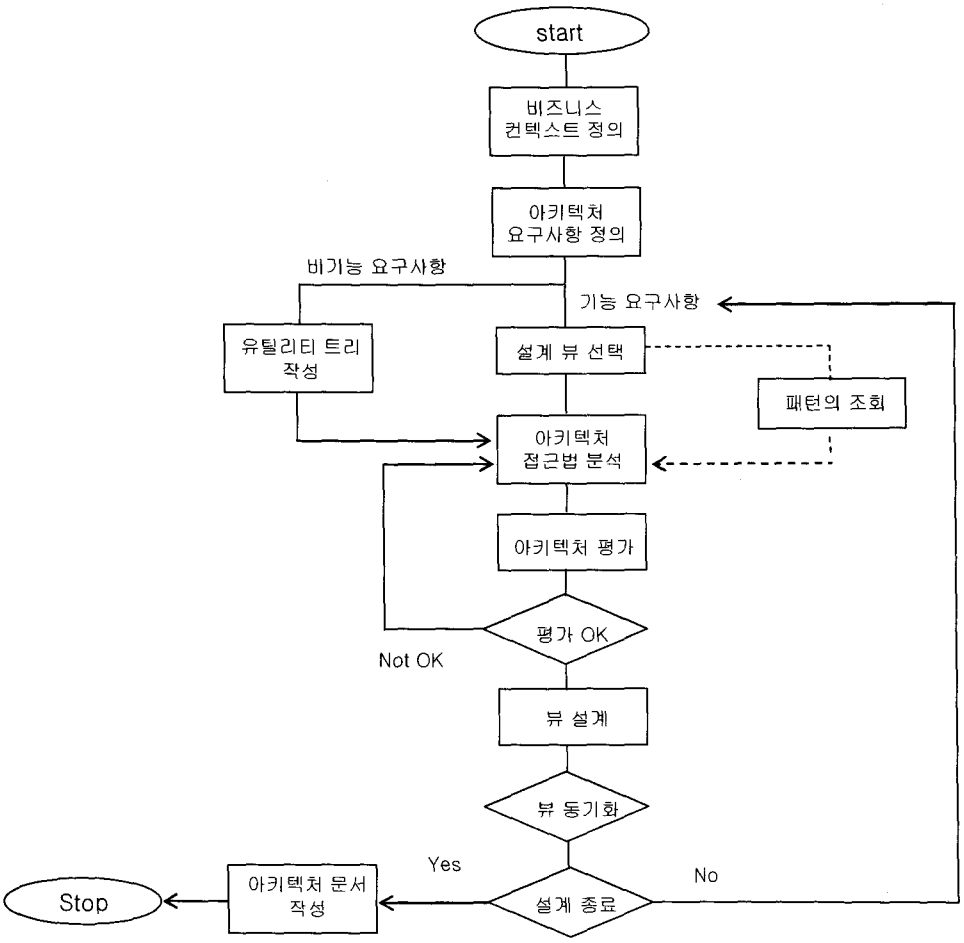


그림 1. 패턴기반 아키텍처 설계절차

4. 패턴의 분류 및 추상화

4.1 패턴의 분류

아키텍처 패턴은 소프트웨어의 유형이 다양한 만큼 최근 들어 매우 많은 새로운 패턴들이 개발되고 잇는 실정이며, 아울러 개발자가 이러한 패턴들을 검색하여 활용하는 것이 중요한 이슈로 대두되고 있다. 따라서 패턴들은 나름대로의 분류체계에 의해 분류되고, 카테고리별로 저장되고, 또한 다양한 방법에 의해 검색이 가능하여야 한다. 패턴의 유형에 따른 분류는 이러한 효율적인 검색에 많은 도움을 준다. 즉, 패턴의 이름만으로 검색한다면 패턴의 이름을 정하는 것이 용이하지 않을 것이며, 원하는 패턴을 찾기가 용이하지 않을 것이다. 패턴의 탐색방법은 결국 패턴을 어떻게 분류할 것인가를 결정해 주는 중요한 요인이 되므로, 논문에서는 기본적인 탐색을 다음과 같이 최초 탐색과 세부 탐색의 2가지 과정을 고려한 아키텍처의 분류와 정의를 수행하고자 한다.

(1) 최초 탐색
- 관련 소프트웨어 자원
- 패턴의 역할
- 아키텍처 뷰
- 관련 소프트웨어 품질요소

(2) 세부 탐색
- 알고리즘 명칭
- 아키텍처 패턴 구성요소
- 키워드 검색

아키텍처 패턴의 분류는 다양한 방법으로 가능하지만 이 분류는 결국 패턴의 검색과 재활용, 또는 확장과 연계

된다는 관점에서 분류가 수행되어야 한다는 점이 중요하다. 디자인 패턴 등의 분류에서와 같이 패턴의 1차적인 분류는 구조적인 성격이냐 혹은 행위적인 성격이냐를 기준으로 분류해 볼 수 있겠다. 또한 다음으로는 아키텍처 뷰(view) 기반의 패턴의 분류를 고려 해 볼 수 있다. 아키텍처에서 가장 중요하게 다루어지는 3가지 뷰 즉, 논리적 뷰(logical view), 동시성 뷰(concurrency view), 배치 뷰(deployment view) 중 어느 뷰에 속하는 가를 통해 분류를 해 볼 수가 있다. 물론 이외에도 다른 관점의 뷰, 예를 들어 신뢰도 측면, 보안 측면의 뷰들로 고려할 수가 있다.

4.1.1 소프트웨어 자원에 기반한 패턴의 분류

우리는 이 외에 아키텍처 패턴의 분류방법으로 아키텍처 패턴이 관여하는 소프트웨어 자원에 의한 분류를 제안해 보고자 한다. 이러한 자원들로는 표3에서와 같이 Software, Momory, CPU, Network, Service, Computer와 같은 요소들이 고려될 수 있다. 이러한 자원들은 다시 세부자원으로 분류되어 패턴을 그룹화 할 수도 있다.

표 3. 소프트웨어 자원에 기반한 패턴의 분류

소프트웨어 자원	패턴과 관련한 자원의 특징	세부 자원	패턴의 유형
Software	모듈 분해		Decomposition Pattern
Memory	공유 메모리	Main Memory Disk Memory 공통	Blackboard
CPU	스케쥴링		Task Scheduling
Network	통신 데이터 전달	Local 통신 Remote 통신 공통	Publish and Subscribe
Service	분산 서비스		Remote Object Access CORBA
Computer	컴퓨팅 노드 분산		Client-Server Web Service 3/5-Tier MVC

4.1.2 패턴의 역할에 따른 분류

각각의 아키텍처 패턴은 패턴의 용도나 역할에 따라 일정한 그룹으로 구분됨을 파악할 수가 있다. 우리는 이러한 관점을 고려하여 아키텍처 패턴을 다음의 9가지 유형으로 구분하였다.

(1) Architecture Style Pattern
- 가장 기본적인 패턴이다. 구체적인 모습이라기보다는 개념적인 모델이다.
- 스타일이라는 용어가 패턴이라는 용어 보다 더 개념적이듯이 본 분류에서도 패턴의 대표적인 의미로 사용한다.
- 다른 패턴들이 여기에 속한 패턴들을 활용하여 구현된다고 볼 수도 있다.

(2) Architecture Decomposition Pattern
- 여기에 속한 패턴의 그룹은 시스템을 분해하는 과정에서 고려되는 패턴이다.
- 논리적 모델의 도메인이나 서브시스템, 서브시스템의 컴포넌트, 그리고 컴포넌트 뷰와 관련한 패턴들이 포함된다.

(3) Publish and Subscribe Pattern
- 하나 이상의 정보 제공자와 하나 이상의 정보 수신자 사이의 전형적인 정보교환 모델을 제공하는 패턴이다.
- 비동기식 정보전달을 위한 모델이 포함된다.

(4) Distributed Service Pattern
- 이 패턴은 복수의 주소공간에 동시에 존재하는 시스템의 구조나, 처리절차, 정책들을 다룬다.
- 분산환경에서 동기식 정보전달 및 서비스 접근을 위한 패턴들이 포함된다.

(5) Service Brokering Pattern
- 분산환경에서상호 연관된 서비스를 명시적인 중재자를 통하여 서로 연계시켜 주는 서비스이다.
- 서비스 통지 및 연결방식에 의해서 다양한 변형을 가질 수 있다.

(6) Memory Management Pattern
- 복잡한 실시간 시스템과 임베디드 시스템을 구축하는 데 있어서의 어려움은 공유되는 자원을 동시에 그리고, 안정되게 관리하는 데 있다.
- 여기서는 전체적인 시스템의 균형을 유지해가면서 메모리와 소프트웨어 자원의 효율적인 관리체 초점을 맞춘다.

(7) Task Scheduling Pattern
- 동시에 처리를 요구하는 작업의 일정계획을 처리하는 패턴들이 포함된다.

(8) Shared Resource Access Pattern
- 메모리 등의 자원에 대한 동시 사용을 통제하기 위

한 패턴들이 포함된다.

(9) Special Purpose Pattern
· 특정한 목적을 위해 사용되는 패턴이다.

4.1.3 패턴의 활용 도메인에 따른 분류

동일한 역할을 수행하는 패턴이라 할지라도, 궁극적으로 그 패턴이 활용되는 분야에 따라 패턴의 규모나 구조나 상호작용은 매우 차이가 있게 되고, 또한 각 패턴이 추구하는 품질속성도 다르게 된다. 패턴의 응용 도메인을 고려한 패턴의 유형에는 다음과 같은 종류가 있다.

(1) 실시간 아키텍처 패턴
(2) 엔터프라이즈 아키텍처 패턴
(3) 분산 서비스 아키텍처 패턴
(4) 웹서비스 아키텍처 패턴
(5) 프로그램언어 종속 아키텍처 패턴
(6) 플랫폼 종속 아키텍처 패턴
(7) 기타 전자상거래 등 응용 종속 아키텍처 패턴

4.2 패턴의 추상화와 관계표현

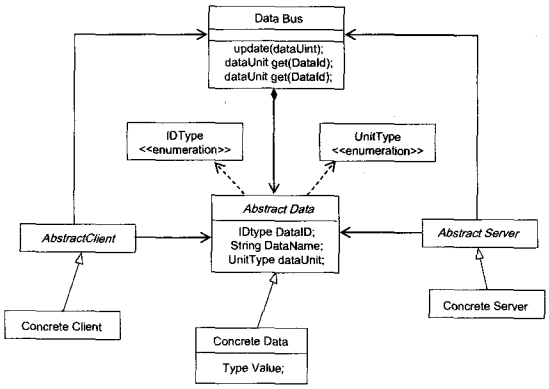
4.2.1 패턴의 추상화

아키텍처 패턴을 표현할 때 만약 패턴의 이해에 주안점을 둔다면 단순한 모델의 표현이 적절할 것이다. 그러나 이 패턴을 이용한 구현을 생각한다면 가능한 상세한 표현이 적합하다고 할 것이다. 그러나 아키텍처는 복잡하고 규모가 큰 소프트웨어이기 때문에, 만약 상세하게 표현된다면 쉽게 이해하는 일이 용이하지는 않을 것이다. 한편, 복잡한 패턴을 단순하고 이해가 용이하게 표현할 때 중요한 것은 표현의 상세화를 어느 정도에서 만족할 것이냐 라는 점에 있다 하겠다. 즉, 세부적인 내용이 아닌 아키텍처 관점에서의 구조와 행위를 나타내는 것으로 충분한 것으로 받아들인다면 추상화 된 패턴을 통해 패턴의 특성을 쉽게 이해 할 수가 있다는 장점을 가질 수가 있다.

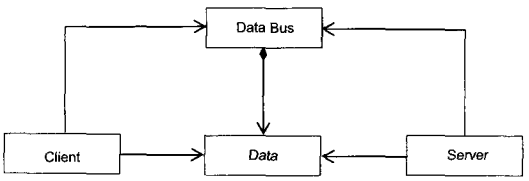
다음의 그림 2의 a)와 b)는 각각 아키텍처 패턴의 상세화된 표현과 단순화된 표현을 동시에 보여 준다. 우리는 단순화된 표현에서 또 다른 기대할 수 있는 장점이 있는데, 그것은 패턴의 변형에 대한 이해와 활용이라고 할 수 있다.

이제까지 정리된 많은 아키텍처 패턴들을 살펴보면, 패턴들에는 유사한 특성이 있으며, 이 특성을 중심으로 패턴들을 그룹화 할 수 있다는 점이다. 또한 이러한 패턴들 간의 유사성 또는 관계로 인해 패턴의 이해나 패턴의 조합이 매우 용이해 지게 된다.

즉, 하나의 패턴과 관련이 있거나 또는 파생된 패턴들은 이러한 패턴들의 기본적인 패턴을 먼저 이해하고, 이 패턴과의 차이나 확장을 이해함으로써 다양한 패턴들을 쉽게 이해할 수 있다. 또한 규모가 큰 소프트웨어의 아키텍처를 많은 관련 있는 패턴들을 조합해서 설계하는 것도 용이해 진다.



a) 상세화 모델



b) 추상화 모델

그림 2. 아키텍처 패턴의 단순화/상세화 표현

4.2.2 패턴간의 관계 표현

아키텍처 패턴을 소개하는 많은 참고문헌에서 소개하는 패턴들은 패턴간 많은 유사성을 가지고 있다. 이것은 하나의 기본적인 패턴을 이해한다면 유사한 많은 변형 패턴들을 이해하는 것이 더욱 쉽다고 하겠다. 우리는 아키텍처 패턴간의 관계를 통해 패턴들의 조합, 변경, 확장을 가능하기 위해서는 패턴 자체의 명세가 유용하다고 생각한다. 패턴과 관련한 그룹의 다이어그램 집합을 여기서 아키텍처 패턴 다이어그램이라고 명명한다. 최근에는 아키텍처나 아키텍처 패턴의 구조를 설명할 때 UML의 다양한 다이어그램이 필수적이지만 아키텍처 패턴 다이어그램은 아키텍처 패턴 자체를 표현하고 패턴간의 관계를 표현하기 위해 별도로 필요하다.

다음의 그림 3은 가장 기본적인 다이어그램으로 특정한 아키텍처 패턴을 나타내기 위한 다이어그램이다. 우리

는 이를 위해 박스와 박스 내에 스테레오 '<<arch pattern>>'이라는 예약어를 사용하고자 한다. 패턴의 이름은 박스 내에 표시한다.

그림 3. 패턴의 다이어그램 표현

아키텍처 패턴들은 상호 관련을 가질 수 있는 예를 들어 하나의 기본적인 패턴은 더욱 상세한 또는 확장된 패턴으로 상속될 수 있다. 다음의 그림 4에서 좌측에 있는 그림은 이러한 상속을 표현하며, 'xyz' 패턴은 'abc' 패턴을 더욱 상세화 하고 있음을 보여 주고 있다. 아래에 있는 그림은 이 내용을 더욱 상세하게 설명하는 데, 'xyz' 패턴이 abc 패턴에 비해 더 확장시키는 부분은 알고리즘 '123'의 부분이다. 이를 통해 두 가지 패턴의 관계는 더욱 명확해 진다. 확장된 내용은 결국 '123'이라는 알고리즘이 추가된 것이라는 것을 알 수 있다. 이러한 예는 구조적으로는 동일한지만 구체적인 알고리즘이나 혹은 내부의 행위적인 측면이 달라지는 패턴들에게 적용될 수 있는 특성이다.

제일 우측의 그림은 기본 패턴에서 확장된 두개의 패턴을 보여 주며, 동시에 이들이 기본 패턴에서 상속되면서 각각 다른 알고리즘을 추가하였을 알 수 있다.

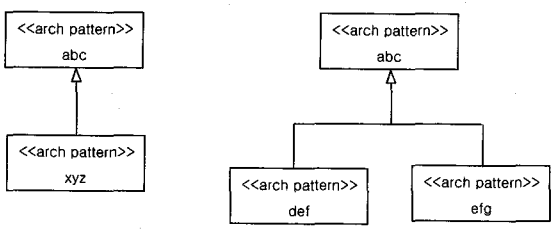


그림 4. 패턴의 상속

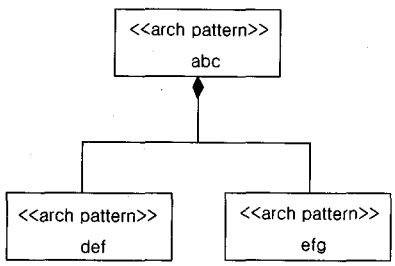


그림 5. 패턴의 집합연관관계

하나의 패턴은 다른 패턴들을 이용해서 만들 수도 있다. 그림 5는 'abc' 패턴이 'def' 패턴과 'efg' 패턴을 통해 구현됨을 보여 준다. 이 패턴의 예는 하나의 종단 간 데이터 전송을 위해서는 통신망에서의 데이터 전송 패턴과 동일 시스템 내에서의 버퍼를 이용한 데이터 전달이라는 두 개의 패턴이 결합됨으로써 가능하게 된다는 점을 생각해 볼 수 있다.

이러한 패턴들 간의 관계정립을 통해 각각의 패턴을 더욱 단순하게 표현할 수 있다는 장점과 다른 패턴과의 관계를 쉽게 이해시킬 수 있고, 더욱이 패턴간의 조합, 변형, 확장 등을 더 체계적으로 수행할 수가 있다는 장점을 제공한다.

5. 품질속성을 고려한 패턴의 정의

5.1 아키텍처를 위한 품질모델

아키텍처에서의 품질모델은 전반적인 소프트웨어의 품질모델인 ISO/IEC-9126이나 IEEE Std 1061-1998, McCall 모델과는 다를 필요가 있다. 기존의 잘 알려진 모델들이 아키텍처의 품질을 포함할 수는 있겠으나, 아키텍처의 설계과정이 이러한 품질을 고려하여 아키텍처 패턴을 선별하고 아키텍처의 설계를 고려한다는 점을 생각해 보면 보다 구체적이고 세분화된 품질모델이 아키텍처의 품질명세에 필요하다고 할 수 있겠다.

이러한 배경으로 본 연구에서는 소프트웨어 아키텍처를 위한 확장된 품질모델을 표 4에 제시한다. 제시된 품질모델은 기존의 품질모델이 소프트웨어의 상위 레벨이나 하위 레벨을 포괄적으로 포함하는 품질을 모델화하여 아키텍처 레벨의 품질을 정의하기에는 부족한 점을 보완하기 위해 아키텍처를 위해 필요한 품질 특성들을 구체화하고 조정한 결과이다. 이 모델의 특징이라면 품질특성 중 안정성과 적응성, 유연성, 구현용이성을 확장한 점을 들 수 있다. 안정성이란 시스템 내의 동작중인 모듈에 대한 통제가 용이한 것을 의미한다. 안정성이 높은 아키텍처는 사용자 명령에 의한 모듈의 개시, 중지 등 통제가 용이하다. 적응성이란 소프트웨어 환경의 변화나 이질적 환경에 대한 신속한 적응 및 수용능력을 의미한다. 유연성은 소프트웨어 환경의 변경에 대해 적응성과는 달리 시스템의 부하를 고려하면서 변경을 용이하게 수용하는 능력을 의미한다. 또한 구현용이성이란 설계된 아키텍처를 상세설계에 옮기거나 프로그램으로 구현할 때 얼마나 용이한가를 표현하는 척도를 의미한다. 아울러, 품질 부특성을 보완한 효율성에는 병렬 및 동시성이나 자원공유성을 추가하여 더욱 세부적인 품질의 명세가 가능하도록 하였다.

표 4. 아키텍처를 위한 품질모델

품질 특성	품질 부특성
기능성	완전성, 정확성, 보안성
신뢰성	무결성, 가용성, 오류허용성
사용용이성	이해성, 학습성, 조작성, 대화성
유지보수성	수정용이성, 확장성, 시험용이성
이식성	하드웨어 독립성, 소프트웨어 독립성, 설치용이성, 재사용성
효율성	시간경제성, 병렬 및 동시성*, 자원경제성, 자원공유성*
안정성(주)1	통제용이성, 균일성(동일한 규격의 플랫폼을 이용)
적응성(주)2	동적 가동성, 호환성, 상호운영성
유연성(주)3	규모성(scalability), 구성 및 변경용이성
구현용이성(주)4	상세설계용이성, 프로그램 용이성

(주)$_{1-4}$: 이 특성들은 보완된 품질 특성 또는 품질 부특성을 표시하고 있음

5.2 패턴의 정의

5.2.1 패턴의 정의항목

패턴의 정의는 관련 연구를 통해 분석한 다양한 정의 자료를 토대로 다음과 같은 항목을 중심으로 정의한다. 이 정의의 특징이라면, 패턴정의는 패턴의 검색을 전제로 하여 패턴의 카테고리가 명확히 명시되고, 또 확장된 아키텍처 품질속성을 토대로 한 품질속성이 정의되며, 아울러 패턴간의 관련성을 위해서는 아키텍처 패턴간의 추상화된 관계를 명시하여 패턴의 이해와 활용이 용이하게 하였다는 점을 들 수 있겠다.

(1) 패턴의 명칭 : 패턴의 이름과 별명을 의미한다.
(2) 패턴의 개요 : 패턴의 개략적인 특징과 적용범위를 간단히 명시한다.
(3) 패턴의 카테고리 : 패턴의 명확한 구분을 위한 유형을 명시한다. 패턴의 유형을 결정하는 소프트웨어 자원 및 패턴이 속한 그룹, 그리고 패턴이 관계하는 아키텍처 뷰를 명시한다.
(4) 패턴의 구조 : 패턴의 구조를 표현하는 다이어그램과 이 다이어그램에 출현하는 구성요소, 그리고 이 구성요소들의 역할을 설명한다.
  · 구조 다이어그램 : 구성요소와 구성요간의 관계를 위한 UML 기반의 도해적인 표현
  · 구성요소 및 역할 : 구조 다이어그램에 나타나는 구성요소들이 패턴을 위한 역할
  · 동적 시나리오 : 아키텍처의 동적인 행위나 상태의 변화를 설명하기 위하여 다이어그램을 이용하여 표현
(5) 패턴의 품질속성
  · 패턴에서 고려하는 중요하고 민감한 품질요소를 중요한 품질속성과 관련된 아키텍처 구성요소를 중심으로 설명함
  · 또한, 특정한 품질요소에 대해 절충(trade-off) 관계에 있는 품질요소도 기술함
(6) 관련 패턴 및 상호 관계
  · 관련 패턴 : 다른 관련이 있는 패턴의 명시 및 관계에 대한 설명
  · 부속 패턴 : 정의된 패턴 내에 존재하는 더 작은 규모의 패턴
  · 알고리즘 : 패턴에 속한 특정한 알고리즘
(7) 구현 고려사항 : 이 패턴을 소프트웨어로 구현할 때 고려해야 할 사항
(8) 적용 사례 : 이 패턴이 활용된 응용사례에 대한 구조와 설명
(9) 참조코드 : 슈도코드

5.2.2 패턴의 정의 사례

앞서 설명한 패턴의 정의항목을 토대로 하여, 이 항목 중 중요한 항목을 중심으로 한 패턴의 정의사례는 다음과 같다.

(1) 패턴의 명칭 : MessageBroker 패턴
(2) 패턴의 개요

임의의 메시지를 전송하고자하는 출판자(Publisher)와 이 메시지의 수신을 원하는 구독자(Subscriber) 사이에 메시지를 중재해 주기 위해 사용되는 패턴이다. 출판자와 구독자 간의 메시지 전달은 메시지 ID를 통해서 구분한다.

(3) 패턴의 카테고리
  · 자원 기반 : Network
  · 패턴 그룹 : Publish and Subscriber 패턴
  · 뷰 : 동시성 뷰

(4) 패턴의 구조

① 다이어그램

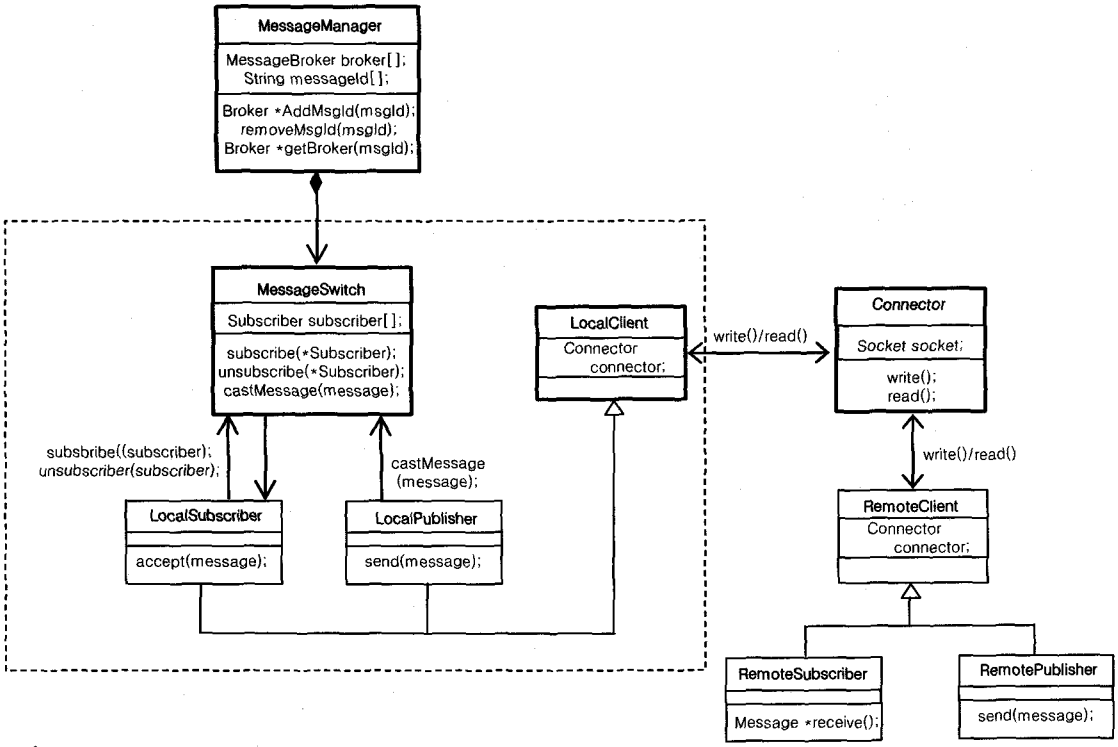


그림 6. MessageBroker 패턴의 구조

② 구성요소 및 역할

o MessageSwitch

- 미리 등록된 구독자들이 원하는 메시지 ID가 들어오면, 그 메시지를 구독하는 구독자들에게 분배한다.
- 구독자들의 구독관리를 위한 함수와 출판자가 메시지를 분배할 때 사용하는 함수가 제공된다.

o RemoteClient

- 메시지를 원격에서 제공하거나 활용하는 클라이언트이다.
- 역할에 따라 출판자와 구독자로 구분된다.

o LocalClient

- 원격에 위치한 클라이언트의 역할을 메시지를 중재하는 서버입장에서 임무를 대행해 주는 클라이언트이다.
- 역할에 따라 출판자와 구독자로 구분된다.

o MessageManager

- 메시지의 교환은 메시지 ID별로 출판자-구독자 그룹에 대해 수행된다. 메시지 관리자는 메시지 ID별로 메시지를 교환해 주는 MessageSwitch를 관리해 준다.

(5) 패턴의 품질속성

표 5. MessageBroker 패턴의 품질속성

품질		관련 구성요소	품질 영향 요인	절충품질
대표 품질	세부품질			
효율능	메시지 전송성능	Connector	병렬 송신 및 수신	안정성, 유연성
	메시지 교환성능	Message Broker	구독자들에 대한 메시지 전달방식	
유연성	클라이언트의 규모 확장에 대한 적응	Message Manager	동일한 관심 메시지에 의한 메시지 처리	효율성
안정성	컴포넌트의 관리	Message Manager	브로커의 관리	효율성
		Message Broker	구독 및 출판자의 관리	효율성

(6) 관련 패턴 및 상호 관계

-이 패턴은 메시지를 교환하는 핵심역할을 하는 MessageSwitch 패턴과 원격에 위치한 메시지 클라이언트와 메시지 서버 사이의 데이터 통신을 담당하는 Channel 패턴으로 구성된다.

-원격의 클라이언트는 크게 메시지 출판자인 Publisher와 메시지 구독자인 Subscriber가 Channel 패턴을 이용한다.

-한편, 서버 측에는 MessageSwitch 패턴의 한 구성요소인 MessageManager가 MessageSwitch 패턴에 포함되어 있다.

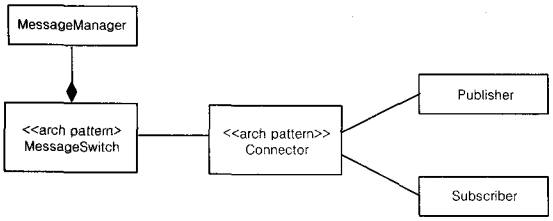


그림 7. Messagebroker 패턴의 관련패턴

6. 평 가

본 연구는 다양한 연구에서 제시되는 아키텍처 패턴을 어떻게 효과적으로 관리하여 아키텍처에 설계에 반영할 것인가에 궁극적인 목적을 두고 있다. 이러한 목적을 달성하기 위해서는 아키텍처 설계방법론을 정립하고, 이 방법론에서 패턴들이 유용하게 활용될 수 있도록 패턴의 분류체계나 정의방법 등을 개선하고자 노력하였다. 또한 이러한 패턴에 대한 체계의 정립은 비단 아키텍처의 설계 뿐 아니라 아키텍처와 패턴을 학습하는 소프트웨어 미숙련자들에게도 패턴에 대한 이해를 용이하게 해 준다는 큰 부대성과를 갖게 해주는 것도 사실이라 하겠다.

본 연구를 통해 수행한 주요 연구내용을 요약하면, 아키텍처 설계방법론의 제안 및 아키텍처 패턴의 분류체계 정립, 패턴의 추상화와 패턴간의 관계표현, 아키텍처 패턴을 위한 품질속성의 확장, 아키텍처 패턴의 명세방법 제시 등이 있으며, 세부 연구 성과 및 기존연구와의 비교는 표 6에 설명되어 있다.

표 6. 연구내용 및 성과분석

연구 내용	연구를 통한 성과	기존 연구의 한계
아키텍처 설계방법론 제안	- 패턴 기반 설계 방법으로 제안함 - 아키텍처의 분석, 설계, 평가 과정을 통합한 절차를 제공함 - 설계 시 아키텍처 품질과 뷰 및 패턴의 적절한 통합방법론으로 제안됨	- 분석, 설계, 평가가 구체적으로 통합되지 못함 - 아키텍처 품질과 아키텍처 뷰에 대한 고려가 통합적으로 이루어 지지 못함
아키텍처 패턴의 분류 체계 정립	- 다양한 아키텍처 패턴의 카타로그를 구축하기 위한 분류체계를 제안함 - 패턴의 탐색을 위한 다각적인 분류체계 도입함 - 소프트웨어 자원이나 패턴의 역할, 패턴의 응용에 따른 분류방법을 제시	- 패턴 유형별로 나름대로의 패턴을 위한 그룹핑이 제안되었으나, 복합적인 패턴들을 수집하여 저장하기 위한 분류체계는 미흡하였음
패턴의 추상화와 패턴 간의 관계 표현	- 복잡한 패턴의 추상화를 통해 패턴의 골격에 대한 이해를 용이하도록 함 - 패턴의 관계를 통해, 하나의 패턴이 포함하는 부속 패턴에 대한 이해를 용이하게 함 - 패턴의 확장 및 조립이 가능하도록 함	- 기존 연구에서 다루어 지지 않은 분야임
아키텍처 패턴을 위한 품질속성의 확장	- 기존의 품질모델을 아키텍처 지향적인 모델로 확장하고 보완하였음 - 특히 소프트웨어의 구조적인 관점을 더욱 구체화하였음	- 국제표준 및 관련 연구 등에서 소프트웨어의 요구사항이나 설계, 구현에 전반적으로 적용 가능한 공통된 품질 모델의 틀을 제시함
아키텍처 패턴의 명세방법 제시	- 확장된 아키텍처 설계방법이나 품질속성, 패턴의 분류체계, 패턴의 추상화 등을 종합적으로 반영한 패턴의 정의방법을 제시함 - 아울러 제안된 항목에 따른 명세사례를 제시함	- 아키텍처 패턴의 유형에 따라 별도의 참고문헌별로 항목을 제시하고 설명함

7. 결 론

이 연구를 통해 발견된 대표적인 연구결과라면 패턴기반 아키텍처 개발방법론과 아키텍처의 분류 및 명세방법, 그리고 아키텍처 패턴의 추상화 방법을 들 수 있다.

이러한 연구결과의 활용방안은 다음의 몇 가지 측면에서 살펴 볼 수 있다. 첫째, 패턴 기반의 아키텍처 개발방법론은 아키텍처 설계에서 패턴을 어떻게 활용하는 가를 제시해 줌과 아울러 설계와 평가를 통합한 현실적이고 유용한 아키텍처 설계방법론으로 활용될 수 있을 것으로 본다. 둘째, 아키텍처 패턴의 명세화 방법은 패턴 각각의 활용도를 제고할 뿐 만 아니라 아키텍처 패턴의 지속적인 연구와 발전을 가속화하는 촉매제 역할을 할 수 있을 것으로 사료된다. 셋째, 아키텍처와 아키텍처의 패턴을 이해하는 데 중요한 자료로 활용될 수 있다. 즉, 아키텍처와 아키텍처 패턴에 대한 이해를 돕는 데 활용될 수 있어 교육적 차원의 기여를 한다고 할 수 있다. 또한 소프트웨어를 개발할 때 아키텍처 설계의 중요한 결정사항을 판단하는 데 또한 유용한 자료로 활용할 수 있다고 본다.

특히, 논문에서는 패턴의 본질에 대한 이해와 확장을 위한 체계를 준비하기 위한 분야를 중점적으로 다루었지만, 향후 구체적인 패턴들을 이러한 체계와 틀 속에서 카타로그로 만들고 데이터베이스를 만들어 가는 작업이 연결되어야 한다고 본다. 물론 이러한 과정에서 분류체계나 정의방법에서 많은 보완 및 개선도 수행될 것으로 본다.

참고문헌

[1] 한국전자통신연구원 연구보고서, 모바일 인터넷 환경에서Dynamic, Scalable 메시지 성능에 관한 연구, 2002.

[2] 한국전자통신연구원 연구보고서, 모바일 인터넷 환경에서 Dynamic, Scalable 메시지 연결에 관한 연구, 2001.

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

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

[5] Felix Bachmann, Len Bass, Gay Chastek, Patric Donohoe, Fabio Perzzi, Architecture Based Design Method, Technical Report CMU/SEI-2000-TR-001, CMU Software Engineering Institute, 2000.

[6] Frank Buschmann, Regine Meunier, Hans Rohnert, Perter Sommerlad, Michael Stal, Pattern-Oriented Software Architecture Volume 1 : A System of Patterns, John Wiley & Sons, July, 2001.

[7] George T. Heineman & William T. Councill, Component- Based Software Engineering-Putting The Pieces Together, Addison-Wesley, 2001.

[8] Jayatirtha Asundi, Rick Kazman, Mark Klein, "Using Economic Considerations to Choose Among Architecture Design Alternatives" Technical Report CMU/SEI, 2001.

[9] Jonathan Adams, Srinivas Koushik, Guru Vasudeva, George Galambos, Patterns for e-business, IBM Press, Oct., 2001.

[10] Jon Siegel and OMG Staff Strategy Group, Developing in OMG's Model-Driven Architecture, Object Management Group White Paper, November, 2001.

[11] Kert C. Wallnau, Scott A. Hissam, Robert C. Seacord, Building Systems from Commercial Components, Addison- Wesley, 2002.

[12] Less Bass and Rick Kazman, Architecture-Based Development, CMU Software Engineering Institute, Technical Report CMU/SEI-99-TR-007, ESC-TR-99 -007, 1999.

[13] Len Bass, Mark Klein, Felix Bachmann, "Quality Attribute Design Primitives and the Attribute Driven Design Method", 4th International Wotkshop on Product Family Engineering Bilbao, Spain, 2001.

[14] Len Bass, Paul Clements, and Rick Kazman, Software Architecture in Practice, 1998.

Mark Klein, Rick Kazman, Attribute-Based Architectural Styles, TM CMU/SEI-99-TR-022, 1999.

[15] Paul Clements, Rick Kazman, Klein, "Evaluating Software Architectures : Methods and Case Studies", Addison Wesley, 2002.

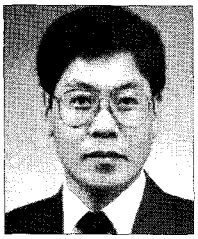
[16] Mark Klein and Rick Kazman, Attribute-Based Architecture Style, Technical Report [18] [19] CMU/SEI-99-TR-022, CMU Software Engineering Institute, 1999.

[17] Rob Wojcik and et al, Attribute-Driven Design(ADD), Version 2.0, Technical Report CMU/SEI-2006-TR -023, CMU SoftwareEngineering Institute, 2006.

[18] Robert T. Monroe, Andrew Kompanek, Ralph Melton, and David Galan, Architectural Styles, Design Patterns, and Objects, IEEE Software, January, 1997.

[19] Yuki Hiroshi 저, 조해미 역 "Java 언어로 배우는 디자인 패턴 입문-멀티 스레드 편" 영진출판사, 2003.

궁 상 환(Sang-Hwan Kung) **[정회원]**



- 1977년 숭실대학교 전자계산학과 졸업(이학사)
- 1983년 고려대학교 대학원 전자정보차리학과 졸업(경영학석사)
- 1998년 충북대학교 대학원 전자계산학과 졸업(이학박사)
- 20001년 ~ 현재 백석대학교 정보통신학과 부교수

<관심분야>
소프트웨어 구조, 분산시스템