論文2002-39TE-3-11

소프트웨어 패턴의 개념적 체계화를 위한 SPO 설계

(A Design of SPO for the Conceptual Systematization of Software Patterns)

洪 顯 述 *, 韓 成 國 **

(Hyeun-Sool Hong and Sung-Kook Han)

요 약

소프트웨어 패턴은 전문가의 검증된 해법과 경험을 토대로 한 문제 해결의 지식표현이다. 그러나, 소프트웨어 개발 문제의 다양성으로 인하여, 현재 발표되어져 있는 수많은 소프트웨어 패턴들 중에서 최적의 패턴을 선택하여 활용하는 것이 용이하지 않다. 이러한 상황은 소프트웨어 패턴의 개념 범주화를 요구하고 있다. 본 논문은 소프트웨어 패턴과 온토로지의 개념 구조를 비교 분석하여, 소프트웨어 패턴이 표출하는 개념 특성을 체계적으로 조직화할 수 있는 소프트웨어 패턴 온토로지(SPO)를 제시하였다. 본 논문에서 제시한 SPO로 소프트웨어 패턴을 개념 수준에서 관리할 수는 기반을 구축할 수 있다. 또한, 본 논문에서는 소프트웨어 패턴과 온토로지 개념을 결합하여 응용할 수 있는 방법을 예시하였다.

Abstract

The software pattern is knowledge representation derived from the verified solutions or the experience of the experts. On account of the design varieties of software development, however, it is not the facilitated task to discover the best proper software pattern. This situation requires that software patterns be categorized in terms of their innate concepts. This paper proposes software pattern ontology(SPO) for the systematic categorization of software patterns by means of conceptual properties of patterns after the comparative analysis of association between software pattern and ontology. The SPO presented in this paper can establish the basis for the software pattern management system at the conceptual level. This paper also shows an idea for the application by unifying conceptual properties of software pattern and ontology.

Keyword : software pattern, ontology, knowledge representation, conceptual categorization.

* 正會員, 圓光保健大學 컴퓨터應用開發科

(Dept. of Computer Application & Development, Wonkwang Health Science College)

** 正會員, 圓光大學校 電氣電子 및 情報工學部

(Dept. of Electrical, Electronics, and Information Engineering, Wonkwang University)

※ 본 논문은 원광보건대학 교내연구비로 연구되었음.

接受日字:2002年1月17日, 수정완료일:2002年8月30日

Ⅰ. 서 론

최근 소프트웨어 공학에서는 반복적으로 직면하게 되는 문제에 대한 성공적인 해결책으로 소프트웨어 패턴(software pattern)이 각광을 받고 있다.[15]

소프트웨어 패턴은 통찰력 있는 전문 소프트웨어 개발자들이 많은 재설계와 재공학 노력을 밑바탕으로 축적한 지식과 경험을 기반으로 도출한 소프트웨어 설계 시 직면하게 되는 제반 문제에 대한 해결 방식이다. 소

프트웨어 개발 전문가들은 소프트웨어의 개발 생산성 향상을 위하여 다양한 디자인 패턴(design pattern)을 발표하고 있으며, 대용량 패턴 데이터베이스가 구축되어 가고 있다. 그런데, 소프트웨어 패턴은 문제의 상황 또는 문맥에 의존적이므로, 일반 개발자가 문제 해결에 적합한 소프트웨어 패턴을 발견하기 위해서는 패턴이 표출하는 개념 특성을 이해하여야 한다. 일반적으로 소프트웨어 패턴에 내재한 개념 특성은 객체 클래스(object class)보다도 추상화되어 있기 때문에, 이를 형식적으로 체계화 하는데 많은 난점이 있다. 소프트웨어 패턴에 내재한 개념 특성에 따라 분류 또는 범주화 하는 방법조차 설정되어 있지 않은 실정이다.

소프트웨어 패턴의 범주화는 패턴 안에 어떠한 개념들이 존재하고, 다른 개념들과 상호 어떤 관계를 형성하고 있으며, 관계는 어떤 개념 의미를 표출해야 하는가를 정의해야 하는데, 특정 도메인의 개념 어휘 체계를 구성하는 온토로지(ontology)는 이를 위한 적합한 접근 방식을 제공하고 있다. 즉, 온토로지의 개념 구조를 이용하여 소프트웨어 패턴에 내재된 개념 특성을 조직적으로 범주화 하는 것이 가능하다. 소프트웨어 패턴을 기술하는 온토로지 기반의 개념 어휘 체계는 패턴의 상위 지식(meta-knowledge) 체계를 형성하여, 패턴의 기술, 검색, 분류 등 소프트웨어 패턴 관리 시스템을 구축하는데 기반 정보를 제공하게 된다.

본 논문에서는 소프트웨어 패턴에 내재한 개념 정보를 온토로지 방법론을 적용하여 분류하고, 온토로지의 어휘를 사용하여 패턴 정보를 기술함으로써 소프트웨어 패턴의 효용성을 제고시키고자 한다.

Ⅱ. 소프트웨어 패턴의 출현과 패턴 분류

1. 소프트웨어 패턴의 출현 배경

소프트웨어 위기를 해결하고자 하는 많은 시도 중에서 1980년대 이후에, 객체 지향 시스템 접근 방식이 각광을 받게 되었다. 객체 지향 시스템은 실세계를 객체(object)라 하는 단일한 개념으로 분석하고 설계하는 기법으로서, 정보 모델링에 새로운 관점을 제시해 주고 있다. 객체지향은 소프트웨어 개발 체계를 개념적으로 접근하는 방식으로, 소프트웨어 위기를 근본적으로 해결할 수 있는 대안으로 인식되어 소프트웨어 개발 체계의 새로운 패러다임으로 정착되었다.[1, 13]

객체지향 패러다임은 시스템 모델링을 위한 해결책을 제시해 주는 것이 아니라, 문제 해결의 접근 방법을 안내하는 사고방식(discipline)이다. 때문에, 객체 지향적 사고방식은 숙달이 어렵고, 소프트웨어 모델링도 난해하다. 이러한 문제점으로 인하여 객체지향 기법 기반의 소프트웨어 개발에서는 개발팀 내에서 프로젝트를 주관하는 숙달된 전문가의 역할이 중요하다. 객체지향 소프트웨어 개발 기법을 효과적으로 적용하기 위해서는 전문가의 경험과 해결방법을 재사용할 수 있는 체계가 필요하며, 이를 명시적으로 구조화한 것이 소프트웨어 패턴이다. 소프트웨어 패턴은 전문가의 해법과 경험을 이해하기 쉽도록 간결한 형태로 기술한 것으로, 소프트웨어 개발자간의 해법과 경험을 교류하기 위한 의사소통 커뮤니케이션 체계이며, 전문가의 검증된 해법과 경험을 바탕으로 한 문제 해결의 지식표현(knowledge representation)이라 할 수 있다.[1, 5, 13]

2. 소프트웨어 패턴의 구성요소

소프트웨어 패턴은 특정한 환경 안에서 다양한 목표와 제약사항들로 이루어진 반복적 문제에 대한 해결책의 명시적 기술서이다. 즉, 소프트웨어 개발 시에 재사용 가치가 있는 경험을 일정한 형식으로 기록한 문서로서, 특정 반복 문제에 대한 해결책을 추상화시킨 개념이다.[4, 14] 소프트웨어 패턴은 내재한 개념 구조를 구조적으로 표현하기 위한 일정한 기술 양식이 필요하며, 이러한 기술 양식은 소프트웨어 개발자간의 의사소통을 가능하게 하는 일종의 공통 언어로서 다음과 같은 요소들로 구성된다.[5]

(1) 명칭 (Name)

소프트웨어 패턴에 부여되는 의미 있는 이름이다.

(2) 문제 (Problem)

소프트웨어 패턴에 정의된 중심 의도와 관련이 있는 문제의 내용을 서술한다.

(3) 환경 (Context)

문제와 해결책이 반복해서 발생하는 상황 또는 소프트웨어 패턴이 적용되기 위한 선행 조건을 기술한다.

(4) 영향력 (Forces)

소프트웨어 패턴의 파급 효과, 부작용과 적용 제약사항 등 소프트웨어 패턴이 활용됨으로써 야기되는 제반 사항을 서술한다.

(5) 해결책 (Solution)

소프트웨어 패턴에서 정의하는 고유의 개념적인 문

제 해결 방안으로 문제 해결을 위한 동적인 규칙과 알고리즘 등을 기술한다. 소프트웨어 패턴의 구조, 패턴의 구성요소와 그 특성, 요소들 사이의 협력 관계를 표현하는 그림, 다이어그램, 또는 문장 등을 포함한다.

(6) 예시 (Examples)

소프트웨어 패턴이 적용되어진 대표적인 예시다.

(7) 결과 환경 (Resulting Context)

소프트웨어 패턴을 적용하여 도출된 결과, 변화되어진 환경, 시스템의 상태 또는 시스템의 형상을 서술한다. 결과 환경에는 새로운 환경으로부터 일어날 수 있는 또 다른 문제점과 연관되는 소프트웨어 패턴들에 대해서도 서술한다.

(8) 이론 근거 (Rationale)

소프트웨어 패턴은 전문가의 경험과 전문적인 통찰력을 바탕으로 생성되는데, 생성된 패턴이 지향하는 목표, 패턴이 생성된 원리, 패턴을 생성한 경험 철학 등 패턴과 관련된 다양한 성찰을 서술한다. 즉, 영향력을 해결하는 이유와 방법에 관한 전체적인 소프트웨어 패턴, 또는 소프트웨어 패턴 내에서의 단계들이나 규칙들의 설명을 정당화시키는 원리이다.

(9) 관련 패턴 (Related Patterns)

같은 유형의 소프트웨어 패턴이나 또 다른 패턴들 사이의 정적, 동적 관련성을 말한다. 관련 패턴들은 영향력을 공유하는 등 공통적인 특성을 갖고 있다.

(10) 사용례 (Known Uses)

기존 시스템에서 해당 소프트웨어 패턴의 응용에 관하여 잘 알려져 있는 사용례를 서술한다.

3. 소프트웨어 패턴의 분류

3.1 소프트웨어 패턴 분류 유형별 비교

소프트웨어 패턴은 문제 해결 방안이나 원리에 대한 고도로 추상화된 기술 형식이다. 따라서, 소프트웨어 패턴을 활용하기 위해서는, 패턴에 내포된 개념 의미를 분명하게 이해하여야 한다. 패턴의 개념 의미를 가시적인 형태로 표출하는 기본적인 방법으로 패턴 분류가 있다. 패턴 분류는 패턴의 개념 의미 특성을 이해하는 출발점이다.

소프트웨어 패턴에 관한 분류방법은 통상 GoF의 분류, Buschmann의 분류, Zimmer의 분류와 Grand 분류의 네 가지로 대별된다.[5,6,7,10,18]

GoF는 패턴이 적용되는 도메인과 패턴이 적용되는 목적을 중심으로 분류하고 있다. GoF의 패턴은 다양한 도메인에서 일반적으로 사용되어질 수 있는 경험들을 추상적으로 표현한 것이므로 적용 영역에 대한 제한은 최소화할 수 있지만, 분류 체계가 포괄적이어서 주어진 설계 상황에서 소프트웨어 설계자를 위해 최적의 분류는 아니다.[5]

Buschmann의 분류는 모든 소프트웨어 패턴의 아키텍쳐를 생성, 구성하기 위해 사용되는 시스템 형성의 빌딩 블록 개념을 기반으로 하고 있다. 이 분류는 소프트웨어 패턴들에 대한 일관성 있는 표현과 소프트웨어 패턴들의 조합에 대한 정보를 포함하여 사용 관점에서의 유용성을 강조하였다. 분류 기준이 되는 입상(granularity)은 응용의 기본 구조에서부터 상세한 구현 부분까지를 포함하기 위한 조건이다. 따라서 소프트웨어 패턴의 크기를 중심으로 한 분류이기 때문에 소프트웨어 패턴이 가지고 있는 특성이나 목적 등에 대한 내용을 수용하지 못하고 있다.[10]

Zimmer는 GoF의 패턴과 그들의 관련성을 기반으로 하여 더 정확한 의미적 정의와 분류를 위해서 새로이 세 가지의 의미적 계층을 추가하여 분류하였지만,[18] Zimmer의 분류는 근본적으로 GoF의 분류와 같은 맥락이기 때문에 소프트웨어 설계자에게 최적의 분류 체계를 제공하고 있지 못하다.

위에서 고찰한 세 가지 분류 방식은 모두 설계패턴만을 대상으로 하고 있기 때문에 소프트웨어 패턴을 수용하는 폭이 좁다.

Grand에 의한 분류 방법은 소프트웨어 개발자들이 해결하고자하는 특정한 문제점과 관련되어 있는 패턴을 좀더 밀접하게 비교·선택할 수 있도록 세분화하여 분류하였고, 또한 소프트웨어 개발 생명 주기인 분석, 설계, 구현, 테스트 등의 전 과정을 대상으로 한 분류 체계로 소프트웨어 패턴 선택의 폭이 넓다는 이점이 있다. 본 논문에서는 Grand에 의한 소프트웨어 패턴 분류를 범주화의 기본 접근 방식으로 하여 소프트웨어 패턴 온토로지 구축에 적용하였다.

3.2 소프트웨어 생명 주기별 패턴 분류

Grand는 소프트웨어 개발 생명주기의 분석, 설계, 구형, 테스트 등의 각 단계마다 발생될 수 있는 문제점의 해결을 위한 분류방식을 취하였다.[6,7]

(1) 기반 설계 패턴(Fundamental Design Patterns)

소프트웨어 설계의 가장 기초적이고 근본이 되는 패턴이다.

(2) 생성 패턴(Creational Patterns)

객체 생성이 요구될 때, 그 객체들을 생성하는 방법에 대한 지침을 제공한다. 인스턴스되는 클래스 또는 객체를 동적으로 결정하며, 결정 과정을 구축하는 방법과 캡슐화하는 방법을 제시한다.

(3) 분할 패턴(Partitioning Patterns)

소프트웨어 생명주기의 분석단계에서 행위자, 개념, 요구사항, 문제를 구성하는 관련성 등을 식별하기 위하여 조사하는 과정에서 복합적인 행위자들과 개념들을 여러 개의 클래스들로 나누는 방법에 대한 지침을 제공한다.

(4) 구조 패턴(Structural Patterns)

다른 유형의 객체들이 하나의 작업을 공동으로 처리할 수 있도록 조직하는 방식을 서술한다.

(5) 행위 패턴(Behavioral Patterns)

행위자를 조직하고, 관리하고, 결합하기 위하여 사용되는 패턴이다.

(6) 병행 패턴(Concurrency Patterns)

병행 연산(concurrent operation)을 조정하는 방법을 제시하는 패턴으로서, 공유되는 자원의 관리와 연산 순서의 적절한 처리 등이 고려할 사항이다.

(7) GRASP(General Responsibility Assignment Software Patterns)

보편적인 객체지향 설계원리를 소프트웨어 패턴의 형식으로 표현하는 것으로 설계 패턴과는 차이가 있다. 제한된 영역 내에 존재하게 될 클래스를 결정하고, 클래스에게 임무를 할당하는 지침을 제공한다. 설계 패턴은 특정한 설계문제를 해결하는 지침을 제공하지만, GRASP은 잘 구조화된 설계에 요구되는 클래스의 적합한 기능에 대한 지침을 제공한다.

(8) GUI 설계 패턴(GUI Design Patterns)

사용자와 친숙한 요소의 사용, 사용자의 기대와 지식베이스가 일치하는 GUI 컴포넌트 설계, 사용자의 실수를 고려하여 수행될 수 없는 연산에 대해서는 경고, 초보자에게는 단계별 기능제공, 경험자에게는 지름길 제공 등을 고려하는 GUI 설계문제를 해결과 관련되는 패턴들이다.

(9) 구성 코딩 패턴(Organizational Coding Patterns)

프로그램 구현시 읽기 및 유지 관리에 쉬운 방법을 제공한다.

(10) 코딩 최적화 패턴(Coding Optimization Patterns)

컴파일러가 자동적으로 최적화를 이룰 수 없는 상태에서, 프로그램의 성능을 개선할 수 있는 방안을 제시한다. 코딩 최적화 패턴의 분야는 반드시 적용을 시켜야 할 정도의 복잡하고 효율적이지 못한 상황에서 사용하는 것이 합리적이다. 그렇지 않으면 프로그램 성능 개선도 가져오지 않을 뿐만 아니라, 오히려 프로그램을 이해하고 유지관리하기를 더욱 어렵게 만들 수도 있다.

(11) 코드 견실성 패턴(Code Robustness Patterns)

더욱 강건한 코드를 작성하는 법을 서술한다. 이 패턴을 사용하면, 소프트웨어 개발비용이 증액될 수도 있다.

(12) 검사 패턴(Testing Patterns)

소프트웨어 테스트에 필요한 다양한 방법을 제시하며, 제한된 제약조건, 테스트되는 소프트웨어의 입상(granularity), 테스트 목적, 테스트 완벽성 정도 등이 고려된다.

Ⅲ. 소프트웨어 패턴 범주화를 위한 온토로지

1. 온토로지와 지식분류

지식베이스는 공통의 지식을 체계화한 지식 저장고일 뿐만 아니라, 구축된 지식베이스의 활용도를 제고하기 위하여서는 지식 공유와 재사용을 위한 에이전트(agent)체제가 구비되어야 하며, 지식 저장고를 보다 빠르게 액세스할 수 있는 방법이 제공되어야 한다. 일반적으로 지식 저장고는 특정 도메인의 지식을 중심으로 구성되는데, 지식의 개념구조화가 선행되어야 한다. 지식의 개념구조는 지식 저장고의 구조를 결정할 뿐만 아니라, 사용자의 지식 저장고 검색을 지원하는 에이전트의 질의어 원형이 되기도 한다. 즉, 지식의 개념구조는 특정 도메인에 대한 상위 지식(meta-knowledge) 이며, 이를 체계적으로 접근하고 있는 것이 온토로지(ontology)이다.[8,9]

온토로지는 개념화에 대한 명확한 명세화이다.[8] 즉, 특정 도메인에서 사용하고 있는 용어(terminology)의 개념 정의와 용어 개념간의 계층적 분류이다. 결과적으로 온토로지는 지식 베이스를 활용하는 다중 에이전트 사이의 커뮤니케이션을 위한 중요한 토대를 형성하며, 다중 에이전트간의 공통 언어로 사용된다.[12]

온토로지에서 지식정보의 수집 및 분류의 기본 방법으로 서로 동떨어진 자료에 대하여 유사한 특징이 있는 것은 묶어 주고 이질성이 있는 것은 분리하는 군집

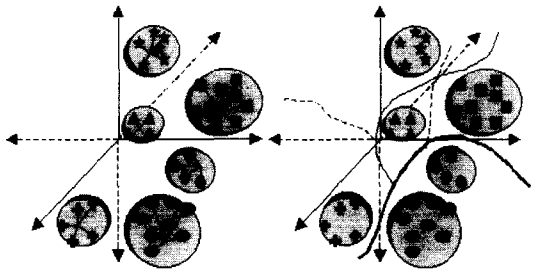


그림 1. 지식정보의 군집 유형
Fig. 1. Clustering of Knowledge Information.

표 1. 온토로지와 소프트웨어 패턴의 비교
Table 1. Comparison between Ontology and Software.

구분	온토로지	소프트웨어 패턴
지식 수준	■특정한 구현에 독립적이며 개념수준에서 지식베이스를 통합하기 위해 기반이 되는 지식구조를 제공 ■이러한 지식은 다중 에이전트들 사이의 커뮤니케이션으로 사용됨	■개념수준에서 단위 지식 요소를 서술 ■소프트웨어 개발에서 자주 도출되는 반복적인 설계 문제에 대한 해결 지식을 효과적으로 서술하기 위한 수단을 제공 ■다양한 프로젝트, 다양한 도메인, 다양한 구현 툴 및 언어에 대하여 개발자들에게 공통적인 방식과 기술을 쉽게 공유하도록 함
핵심 내용	■개념 요소의 계층적 관계가 중요시 됨 ■개념요소는 의미, 목표, 기능, 기능성, 컴포넌트, 구조, 특성 등을 내포 ■개념요소 사이의 계층이나 상속 관계는 개념 클래스 또는 범주 네트워크 형태로 표현됨 ■개념 클래스 또는 범주 네트워크 구축시에는 도메인 관련 지식이 중요한 역할을 하며, 도메인 관련 지식으로는 개념들 사이의 관련성(예를 들어, part-of, 또는 instance- of), 개념 특성(property), 제약조건(constraint), 어휘 확장 규칙 등이 활용됨 ■특정 도메인에서의 지식베이스 구축에 필요한 개념 구조를 정의	■다양한 환경에서 분석적, 설계적, 구조적인 문제들에 대해 효율성이 입증된 해결책 ■문제-해결책의 전문지식을 관련 요소간의 관계를 명확하게 표현 할 수 있는 그래픽 표기를 활용하여 도식화 함 ■본질적 특성은 특정한 상황하에서 지정된 문제에 대한 해결 방안과 이 해결방안의 합정, 힌트, 실제로 소프트웨어 패턴 적용시 예측되는 장단점과 긍정적인 결과 등을 기술하는데 있음 ■소프트웨어 개발의 일반적인 문제에 대한 전문 지식으로서 다양한 도메인에 확대 적용할 수 있음
상관성	■도메인 지식을 바탕으로 하향식(top-down)으로 구성됨 ■새로운 소프트웨어 패턴의 생성 또는 존재를 예측할 수 있음	■개별 패턴의 상향식(bottom-up)으로 구성됨 ■소프트웨어 공학 온토로지의 일부분으로써 온토로지 내의 개념요소에 대한 또 다른 구체적인 명세 방식이라 할 수 있음 ■온토로지의 어휘 개념에 대응되며, 소프트웨어 패턴의 명세 구조는 온토로지 어휘 개념의 명세 구조로 간주 할 수 있음 ■온토로지의 분류와 계층구조 형성에 중요한 개념을 제공
어휘와 의미	■특정 도메인에서 사용되는 어휘의 선언적 정의로 도메인 개념의 명세화 ■어휘개념 정의에는 도메인 지식, 제약조건, 개념 간의 관련성, 도메인 개념의 계층구조가 포함됨	■소프트웨어 개발 대안을 탐색하고, 개발 기법을 문서화하고, 개발자 상호간의 의사소통을 위한 공통어휘를 제공 ■서술형식과 사용되는 어휘들이 비형식적이지만, 상위레벨의 소프트웨어 공학지식을 효과적으로 활용하며, 개념의 추상화를 제공함으로써 시스템 개발의 복잡도를 감소시킴 ■소프트웨어 개발 과정에 사용되는 어휘 개념에 대한 공학적인 의미를 기술
분류와 구조	■도출된 개념 범주가 갖고 있는 일련의 속성들에 대한 'is-a' 관계 중심의 계층 구조 형성 ■속성들 사이의 'part-whole' 등과 같은 관계는 계층 구조의 내부에 포함되어 표현됨 ■개념 분류를 기반으로 하여 개념 관계를 계층적 트리구조로 형성함 ■분류에 중점을 둠	■도메인 내에서 발생하는 문제에 대한 해결 방안을 명시적 형태로 구조화 함 ■패턴 고유의 의미 개념과 이와 연관되는 클래스 또는 객체의 관계를 명세화 함 ■구조 형성에 초점을 맞춤

방법(clustering)이 있다. 군집 방법에는, 그림 1처럼 이미 정해진 그룹과 새로운 개체를 묶어주는 응집 방법과, 하나의 큰 부분을 세부적으로 나누어 가는 분할 방법이 있다. 온토로지에서는 이렇게 응집과 분할을 반복해 가며 의미적 관련성이 있는 개념을 추출하여 지식의 체계화를 달성한다.

지식정보의 군집화에 의해 생성된 온토로지는, 에이전트가 개념을 이해하고 정보를 수집하는데 지침이 되며, 수집한 정보는 온토로지의 유형에 따라 체계적으로 분류·정리된다. 온토로지는 분류된 정보를 접근하기 위한 색인이 되며, 사람·기계 공통의 어휘를 제공함으로서 인간과 에이전트에 의한 공동작업을 가능하게 한다.[9]

2. 온토로지와 소프트웨어 패턴 비교

온토로지는 지식베이스 구축에 필요한 기본 구조나 골격을 제공한다.[17] 반면에, 소프트웨어 패턴은 실제적인 상황(context)에서 이용되어 왔던 것으로, 또 다른 상황에서 재사용 될 수 있는 지식을 의미한다.[3] 온토로지와 소프트웨어 패턴은 개념 구조를 형상화는 관점에서 밀접한 유사성이 있으며, 상호 보완적인 독자적 특성을 내포하고 있다.

지식베이스 시스템 구축에 응용되고 있는 온토로지는 개념 구조화를 기반으로 하는 객체지향 소프트웨어 시스템에도 적용될 수 있다. 소프트웨어 시스템은 개발하고자 하는 도메인의 지식을 바탕으로 설계 모델을 구성하기 때문에, 온토로지는 설계도메인의 지식 표현과 지식 공유의 공통 언어로써 효과적인 수단이 된다. 온토로지는 설계 도메인에 내포되어 있는 지식을 추출하고 이들 사이의 관계를 제공함으로써 소프트웨어 설계 모델 구성에 결정적인 역할을 할 수 있다.[2]

온토로지가 도메인 지식을 표현한다면, 소프트웨어 패턴은 소프트웨어 설계에 필요한 단위 기반 지식 표현 구조이다. 온토로지와 소프트웨어 패턴은 지식 표현의 한 방법이라는 유사성을 공유하고 있으며, 상대적 특성을 비교분석하면 표 1과 같다.

표 1의 온토로지와 소프트웨어 패턴의 비교 분석을 종합해 보면, 온토로지와 소프트웨어 패턴은 지식 공유와 지식 재사용에 대한 유사한 접근 방식과 목적을 갖고 있다. 개념 형성, 개념간의 관계설정, 특정 도메인 내에서의 공유가 가능한 어휘 개념과 공통 언어, 개념 수준에서의 지식 표현 등 온토로지와 소프트웨어 패턴은 개념 표현에 있어 유사한 목표 실현을 목적으로 한

다. 또한 온토로지와 소프트웨어 패턴은 특정 도메인의 본질적 개념 구조 표현을 추구하고 있으며, 객체지향 개념을 바탕으로 전개되고 있다.

그러나, 온토로지와 소프트웨어 패턴은 도메인의 성격에 따라 상이하게 나타나고 구현 방식이나 기초 이론에 있어서도 상이한 차이가 있다. 때문에 특정 도메인의 보다 수준 높은 개념 구조 표현을 위해서는 온토로지와 소프트웨어 패턴이 상호 보완적인 역할을 하여야 한다.

3. 온토로지의 생성

온토로지 기반의 지식관리 시스템은 특정 도메인 내에서 공유되고 있는 어휘를 사용하여 전체 도메인의 담화구조(discourse)를 정의한다. 따라서 온토로지는 특정 도메인에 표출될 수 있는 모든 담화구조를 수용하기 위하여 작은 어휘 개념 단위로 세분화되어야 한다.[12] 온토로지는 개념화된 어휘와 어휘관계를 계층적 형태로 분류하며, 이를 의미 네트워크(semantic network) 형태의 그래프나, 계층적 트리구조, 체계적인 서술식 텍스트형식 등으로 표현한다. 표현 형식은 개념을 이루는 어휘와 어휘 정의를 주축으로 구성되지만, 객체, 범주(category), 관계, 추론 규칙(inference rule), 특성, 속성, 제약조건, 기능, 상수(constant) 등 다양한 요소가 동시에 서술된다.[16]

온토로지 생성 방법으로는 구축하고자 하는 도메인의 개념을 자연언어로 기술하여 분석하는 방법 등 여러 방법이 제안되고 있지만, 알고리즘에 따라 동작하는 컴퓨터 시스템에 적합한 온토로지를 구성하는 것은 어렵다. 표 2에 온토로지 개발에서 야기되는 문제점을 요약 정리하였고, 그림 2에는 온토로지 기반 시스템의 일반적인 구성도를 보였다.[19,20]

표 2. 온토로지 개발의 문제점
Table 2. Problems of Ontology Development.

하향식 개발	온토로지를 사용하는 방법에 따라 추가, 변경이 어렵다.
다중성의 배제	동일개념이 한 방법으로 밖에 정의되지 않거나 상황에 따라 다양한 형식의 기술이 가능하지 않다.
맞춤 생성 난해	각자의 의견을 반영시키는 방법에 따라, 개별적으로 이용 가능한 온토로지를 생성할 수 있다.
감독 기능 미비	온토로지 사용자의 온토로지 이용상황을 평가(monitoring)하기 어렵다.

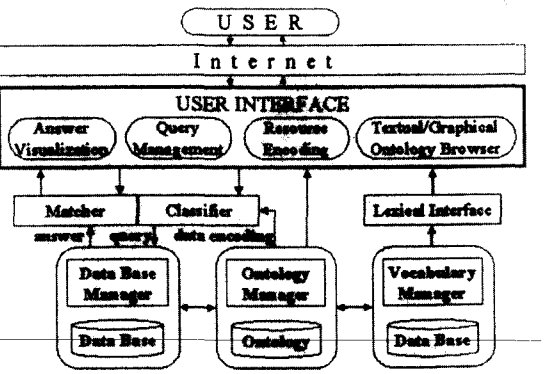


그림 2. 기본적인 온토로지 기반 시스템
Fig. 2. Typical Ontology-based System.

온토로지를 이용함으로써 얻을 수 있는 이점으로는 개념이 다른 두 요소 사이의 인터페이스를 위한 맵핑(mapping) 설계가 용이하고, 온토로지의 상위지식은 질의 처리에서 지능형 검색이 가능하도록 하여 주는 것 등이 있다. 그리고 다른 시스템에 이미 존재하고 있는 개념 요소를 전부 또는 선별적으로 재사용할 수 있으며, 온토로지 부트스트래핑(ontological bootstrapping)으로 새로운 온토로지를 구축하는데 요구되는 소요시간을 단축할 수 있다.[11]

Ⅳ. 소프트웨어 패턴 온토로지(SPO) 설계

1. 소프트웨어 패턴 지식의 분류

온토로지는 지식관리 시스템에서 사용자와 지식베이스간의 커뮤니케이션을 위한 개념 수준의 공통 어휘를 제공하여, 의사소통의 담화를 형성할 수 있게 한다. 온토로지는 보다 정밀하고 정확한 의사소통을 위해 어휘 개념을 세밀한 수준으로 세분화해야 한다.[21] 따라서, 소프트웨어 패턴 온토로지를 추출하기 위해서는 먼저 소프트웨어 패턴을 개념적인 측면에서 세부적으로 분류하여야 한다.

(1) 소프트웨어 패턴의 개념적 분류

소프트웨어 패턴은 소프트웨어 설계 과정을 고도화하고, 기존의 검증된 지식을 재사용하여 생산성을 제고하는데 그 목적이 있다. 때문에, 소프트웨어 패턴은 소프트웨어 개발 과정과 연동될 때 진가를 발휘할 수 있다. 본 논문에서는 소프트웨어 개발 주기 중심의 Grand 분류법을 기반으로 하여 소프트웨어에 패턴을 개념적

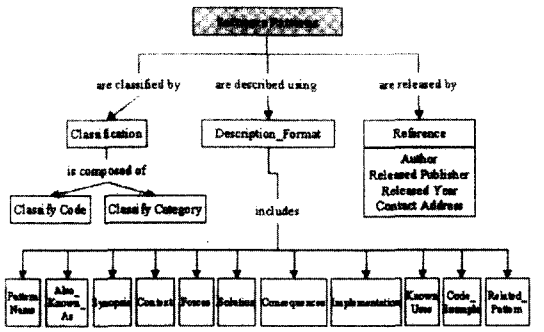


그림 3. 소프트웨어 패턴 지식의 상위 구조
Fig. 3. Higher System of the Knowledge of Software Pattern.

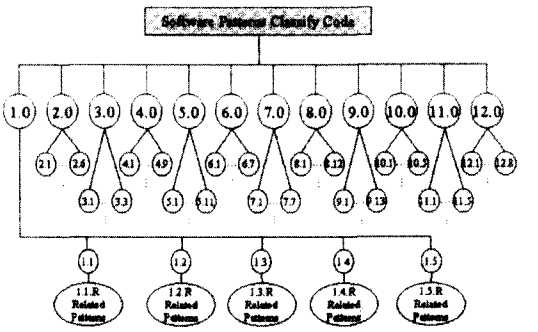


그림 4. 소프트웨어 패턴 분류코드의 트리구조
Fig. 4. Tree of the Classify-Code of Software Pattern.

으로 분류하는 방법을 제시한다. 그림 3은 본 논문에서 제시하는 소프트웨어 패턴의 상위 범주 구조를 보인 것이다.

소프트웨어 패턴을 먼저 세 가지 항목으로 대별한다. 소프트웨어 패턴이 소프트웨어 생명주기에서 어느 단계의 어떤 기능을 해결할 수 있는 것인가를 선언하는 분류(classification) 항목, 소프트웨어 패턴을 지정된 형식으로 기술하기 위한 포맷(format)항목, 그리고 소프트웨어 패턴에 대한 참조(reference) 항목의 세 항목으로 구성된다.

분류항목은 다시 분류코드(classify-code)와 분류 카테고리(classify category)로 나눈다. 참조 항목은 패턴을 발견하여 발표한 사람, 소프트웨어 패턴이 공식적으로 발표된 출판물과 발표 연도 등을 필요에 따라 선택적으로 사용한다.

소프트웨어 패턴 작성을 위한 포맷 항목을 Name(명칭), Also Known As(별칭), Synopsis(개요), Context(환경요인), Forces(고려사항), Solution(해결책), Consequences(결과), Implementation(구현), Known Uses(활용예), Code Example(코딩예), Related Patterns(관련패턴) 등의 요소들로 구성한다.

(2) 소프트웨어 패턴 분류 코드 생성

소프트웨어 패턴 온토로지에서 패턴 범주의 효율적인 관리를 위하여, 소프트웨어 패턴 범주와 그 하위에 속한 소프트웨어 패턴에 각각 계층 레벨을 기준으로 분류코드를 부여한다.

본 논문의 소프트웨어 패턴 온토로지 SPO에 등록된 소프트웨어 패턴은 그림 4와 같이 분류한다.

소프트웨어 패턴의 분류 범주는 12가지 유형으로 분류되므로 1.0에서부터 12.0까지 순차적으로 코드를 부여한다. 각각의 범주 안에 포함되는 패턴들은 계층구조를 의미할 수 있도록 하위코드를 부여한다. 예를 들어, 패턴 범주 1.0(Fundamental Design Patterns) 안에는 모두 5개의 소프트웨어 패턴이 등록되어 있는데, 이들은 각각 1.1(Delegation), 1.2(Interface), 1.3(Immutable), 1.4(Marker Interface), 1.5(Proxy) 등으로 분류코드를 부여한다. 또한 각 패턴의 분류코드의 하위코드로 '~.R' 을 덧붙여서 관련 패턴들의 목록을 참조할 때 활용되도록 한다.

특히 패턴 분류코드는 소프트웨어 패턴 검색 과정에서 관련된 패턴의 유사성 정도의 기준을 정하는데 활용할 수 있다. 즉, 그림 5의 예와 같이 같은 범주내의 패턴은 다른 범주의 패턴들보다 상대적으로 유사도가 더 높게 취급된다. 이러한 방식으로 본 논문에서 분류된 전체 소프트웨어 패턴의 수는 그림 4에서 보는 바와 같이 총 91개이다.

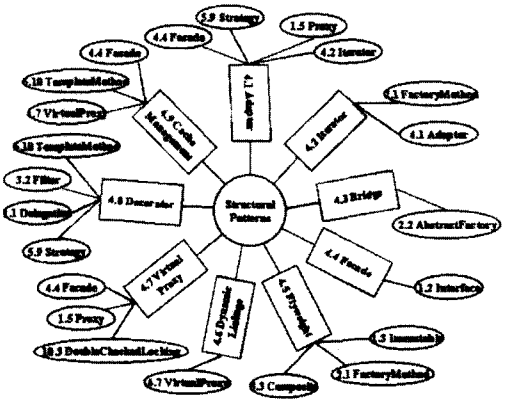


그림 5. 구조 패턴의 분류코드 관계도
Fig. 5. Relationship of the Classify Code of Structure Patterns.

(3) 소프트웨어 패턴 범주

본 논문에 등록된 소프트웨어 패턴 범주는 그림 6과 같다.

2. 소프트웨어 패턴의 온토로지 작성

본 논문에서 작성한 SPO는 소프트웨어 패턴 지식의 구조적인 세분화 작업의 기준이 되는 범주 부분, 패턴 지식에서 추출된 개념과 그 개념의 의미를 설명하는 정의(definition) 부분, 그리고 추출된 개념간의 관련성(relationship) 부분이 주요 핵심 구성 요소이다. 이외에도 SPO는 객체 또는 개념들의 명칭을 재명명(rename)하는 부분과 패턴 분류코드 부분 등으로 구성된다. 각 부분의 세부 내용은 다음과 같다.

(1) 소프트웨어 패턴 범주 부분

표 3은 본 논문에서 작성한 소프트웨어 패턴 온토로지의 개념 범주 집합을 계층적 구조로 표시한 것이다. ()는 개념의 속성을 나타낸다.

(2) 정의(Definition) 부분

소프트웨어 패턴 지식에서 추출한 키워드와 소프트웨어 패턴의 기초 이론을 제공하는 객체지향 시스템의 이론적 개념들에서 추출된 어휘들의 목록을 작성하고, 그 어휘가 의미하는 개념구조를 서술한다. 정의부분은 기능에 따라 다음 세 부분으로 나누어진다.

- 객체지향 시스템의 기초개념 및 각 패턴 내에 내포되어 있는 공통 어휘를 정의하는 부분(Common_Definition)
- 소프트웨어 패턴 온토로지에서 선언된 범주 어휘를 정의하는 부분(Category_Definition)
- 속성을 정의하는 부분(Attribute_Definition)

그림 7은 소프트웨어 패턴 온토로지 어휘의 작성 예이다.

(3) 관련성(Relationship) 부분

SPO에 등록된 객체 또는 개념들간의 관계를 서술한다. 관련성 표기의 형식은 다음과 같이 관계식 표현을 사용한다.

$$<Ci> <<R>> <Cj>$$

여기서 Ci, Cj는 객체 또는 개념, R은 관련성이다.

관련성에는 여러 형태가 있는데, 그림 8에서는 소프트웨어 패턴에 공통적으로 관계되는 관련성, 패턴 내용을 작성할 때 작성 요소간의 관련성, 패턴 분류에 관계된 관련성, 그리고 각각의 패턴이 또 다른 패턴들과 관

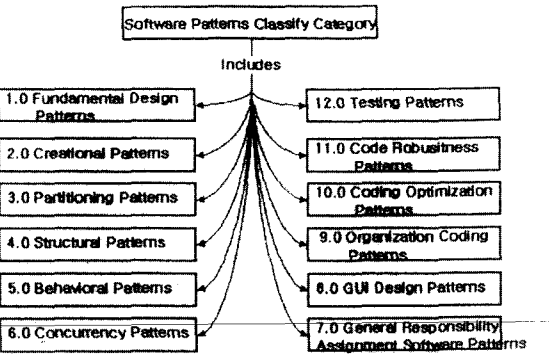


그림 6. 소프트웨어 패턴의 범주
Fig. 6. Category of Software Pattern.

표 3. 소프트웨어 패턴 온토로지 범주
Table 3. Category of Software Pattern Ontology.

Software Patterns
Classification
Classify_Code
Classify_Category
Common Attributes of Software Patterns : (Generalization, Specialization, Inheritance, Aggregation, Delegation)
Fundamental Design Patterns
Creational Patterns
Partitioning Patterns
Structural Patterns
Behavioral Patterns
Concurrency Patterns
GRASPs
GUI Design Patterns : (Experience Level of user)
Organizational Coding Patterns
Coding Optimization Patterns
Code Robustness Patterns
Testing Patterns : (Condition, Granularity, Thoroughness)
Reference : (Author, Released Publisher, Released Year, Contact Address)
Elements of Pattern Description
Name
Also Known As
Synopsis
Context : (Problem, Precondition)
Forces : (Force, Constraint)
Solution : (Dynamic rule, Static relationship)
Consequences : (Good and bad result, Resulting context)
Implementation : (Pitfall, Hint, Technique)
Known Uses
Code Example : (Programming Language)
Related Patterns : (Relationship, Difference)

계를 형성 할 때의 관련성 등의 작성 예를 보였다.

(4) 재명명(Rename) 부분

재명명 부분은 객체의 이름이나 개념을 나타내는 용어에 대해서 사용상에 융통성을 부여하기 위하여 작성한다. 본 논문에서 사용하는 명명규칙은다음과 같다.

첫째, 패턴의 고유 이름은 하나의 단어로 명명된 것도 있지만, 여러 개의 단어들이 모여 명명되는 경우도 있다. 이러한 경우, 각 단어의 첫 글자를 대문자로 하고 나머지는 소문자로 하여 단어 사이에 공백이 없이 표기한다.

```
1. Common_Definition

Object : 객체를 나타내는 데이터(data)와 데이터 연산을 위
         한 기능(function)을 캡슐화한 기본 개체
Class : 객체를 생성하는 템플리트
Method : 캡슐화된 데이터 조작
Message : 객체의 상태 또는 리턴값을 변경하기 위하여 객
          체에게 보내는 요청
Abstraction : 추상회된 특성, 개념과 사고를 나타내는 것
Encapsulation : 객체의 세부 사항들을 은폐하는 과정
                        :

2. Category_Definition

Classification : 패턴의 분류
Classify_Code : 패턴분류를 토대로 하여 계층적 구분이 가
                능하도록 부여한 코드
Classify_Category : 패턴분류의 범주
Fundamental Design Patterns : 설계에서 기초가 되는 패턴
Creational Patterns : 1)객체들을 생성하는 방법에 대한 지
    침을 서술하는 패턴
    : 2) 인스턴스하는 클래스와 임무를 위임받을 객체들을
    동적으로 결정하고 캡슐화하는 방법을 제시하는 패턴
Partitioning Patterns : 분석단계에서 복합적인 행위자들과
    개념들을 여러 개의 클래스들로 나누는 방법에 대한
    지침을 서술하는 패턴
                        :

3. Attribute_Definition

Concur_Pattern.Shared_Resource : 병행 패턴에서의 공유자원
Concur_Pattern.Sequence_of_Operation : 병행 패턴에서의 공유
                                       자원의 처리순서
Test_Pattern.Condition : 테스트하기 이전의 미리 통제된 조건
Test_Pattern.Granularity : 테스트할 소프트웨어 입상규모
Test_Pattern.Thoroughness : 테스트의 완벽성 정도
Reference.Author : 패턴을 발견하여 발표한 사람
Context.Problem : 패턴이 해결해야할 문제점
                        :
```

그림 7. 소프트웨어 패턴 온토로지 어휘 정의 예
Fig. 7. Examples of the Words Definition of Software Pattern Ontology.

```
++++++++++++++++++ 소프트웨어 패턴의 공통적인 관련성 ++++++++++++++++++++

<Software_Patterns> <<are classified by>> <Classification>
<Classification> <<is composed of>> <Classify_Code>
<Classification> <<is composed of>> <Classify_Category>
<Classify_Category> <<includes>> <Fundament_Design_Pattern, Creation_Pattern,
                                  Partition_Pattern, Structure_Pattern, Behavior_Pattern,
                                  Concur_Pattern, GRASP, GUI_Design_Pattern,
                                  Organization_Coding_Pattern, Coding_Optimiz_Pattern,
                                  Code_Robust_Pattern, Test_Pattern>
<Software_Patterns> <<are released by>> <Reference>
<Reference> <<are>> <Ref_Author, Ref_Publisher, Ref_Year, Ref_Contact_Address>
<Software_Patterns> <<have been discovered by>> <Ref_Author>
<Software_Patterns> <<are described using>> <Description_Format>
<Description_Format> <<includes>> <Name, Also_Known_As, Synopsis, Context,
                                   Forces, Solution, Consequences, Implementation, Known_Uses,
                                   Code_Example, Program_Language, Related_Patterns>

++++++++++++++++++ 패턴 작성의 구성요소에 관한 관련성 ++++++++++++++++++++

<Name> <<is>> <Name_of_Pattern>
<Name> <<will become>> <Part_of_Design_Vocabulary>
<Name> <<is described using>> <Characters>

<Also_Known_As> <<is>> <Alias_of_Pattern_Name>
<Also_Known_As> <<is described using>> <Characters>

<Synopsis> <<is>> <Abstract_of_Pattern_Description>
<Synopsis> <<is described using>> <Text>

<Context> <<describes>> <Problem_of_Pattern>
<Context> <<suggests>> <Design_Solution_to_Problem>
<Context> <<is described using>> <Content>
                                        :
++++++++++++++++++++ 패턴 분류에 관한 관련성 ++++++++++++++++++++++++

<Fundament_Design_Pattern> <<are>> <Delegation, Interface, Immutable,
                                    MarkerInterface, Proxy>

<Creation_Pattern> <<are>> <FactoryMethod, AbstractFactory, Builder, Prototype,
                            Singleton, ObjectPool>

<Partition_Pattern> <<are>> <LayeredInitialization, Filter, Composite>

<Structure_Pattern> <<are>> <Adapter, Iterator, Bridge, Flyweight, DynamicLinkage,
                             VirtualProxy, Decorator, CacheManagement>

<Behavior_Pattern> <<are>> <ChainOfResponsibility, Command, LittleLanguage,
                            Mediator, Snapshot, Observer, State, NullObject,
                            Strategy, TemplateMethod, Visitor>
                                        :
                                        :
++++++++++++ 각각의 패턴과 관계되는 다른 패턴들과의 관련성 ++++++++++++++

+++++++++ <Fundament_Design_Pattern>과 관계되는 패턴들의 관련성 +++++++++

######### <Related_Patterns_to_Delegation>
<Almost_Every_Other_Patterns> <<use>> <Delegation>
<Decorator> <<relies on>> <Delegation>
<Proxy> <<relies on>> <Delegation>

######### <Related_Patterns_to_Interface>
<Interface> <<is used with>> <Delegation>
<Interface> <<is used by>> <Many_Other_Patterns>

######### <Related_Patterns_to_Immutable>
<Immutable> <<is used to avoid access-coordination of>>
                                        <SingleThreadedExecution>
<Immutable> <<is used to avoid>> <Other_Kind_of_Access_Coordination>
                                        :
                                        :
```

그림 8. SPO에 등록된 객체 또는 개념들간의 관련성 예
Fig. 8. Examples of the Relationships between Objects or Concepts in SPO.

둘째, 패턴 이름을 제외한 나머지 객체 또는 개념의 명칭은 단어와 단어 사이에 밑줄(_)을 이용하여 연결시킨다.

셋째, 너무 길게 여겨지는 단어는 의미를 손실하지 않는 범위에서 축약어로 표기한다.

넷째, 여러 개의 단어들이 모여 하나의 개념을 나타내는 명칭은 의미를 손실하지 않는 또 다른 단어를 사용하여 간단하게 표기를 한다. 예를 들면, 'Elements of Pattern Description'의 재명명을 'Description_Format'과 같이 한다.

(5) 패턴의 분류코드(Classify Code) 부분

패턴 분류 범주와 각각의 패턴들에는 분류코드를 부여한다. 표 4는 부여된 소프트웨어 패턴 분류코드의 일부분을 계층적으로 보인 것이다.

3. SPO 구축

소프트웨어 패턴 지식의 분류 과정을 통해서 SPO를 생성할 수 있다. SPO는 소프트웨어 패턴 지식에 온토로지 이론을 적용하여 구축한 소프트웨어 패턴 지식정보의 범주화 체제이다. 그림 9는 본 논문에서 구축한 SPO의 소프트웨어 패턴 범주가 계층구조로 체계화된 형상을 보인 것이다.

또한 총 91개의 샘플링된 소프트웨어 패턴을 적용하여 SPO로부터 추출된 개념(concept)은 150개, 관계(relation)는 120개이다. 그리고 구축된 소프트웨어 패턴

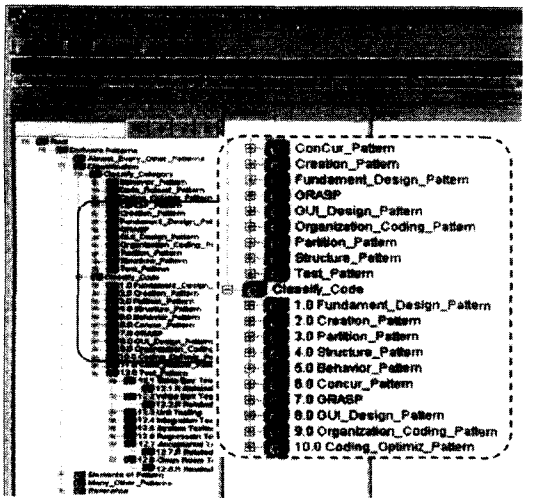


그림 9. SPO의 소프트웨어 패턴 범주 계층구조
Fig. 9. Hierarchy of the Software Pattern Category in SPO.

표 4. SPO의 패턴 분류 코드 작성 예
Table 4. Examples of the Classify Code of Software Pattern in SPO.

1.0	Fundament_Design_Pattern		
	1.1	Delegation	
		1.1.R	Related_Patterns to Delegation
	1.2	Interface	
		1.2.R	Related_Patterns to Interface
	1.3	Immutable	
		1.3.R	Related_Patterns to Immutable
	1.4	Marker Interface	
		1.4.R	Related_Patterns to MarkerInterface
	1.5	Proxy	
		1.5.R	Related_Patterns to Proxy
2.0	Creation_Pattern		
	2.1	Factory Method	
		2.1.R	Related_Patterns to FactoryMethod
	2.2	Abstract Factory	
		2.2.R	Related_Patterns to AbstractFactory
:	:	:	:
	11.4	Return New Objects from Accessor Method	
		11.4.R	Related_Patterns to RNOFAM
	11.5	Copy Mutable Parameters	
		11.5.R	Related_Patterns to CopyMutableParameters
12.0	Test_Pattern		
	12.1	Black Box Testing	
		12.1.R	Related_Patterns to BlackBoxTesting
	12.2	White Box Testing	
		12.2.R	Related_Patterns to WhiteBoxTesting
	12.3	Unit Testing	
		12.3.R	Related_Patterns to UnitTesting
	12.4	Integration Testing	
		12.4.R	Related_Patterns to IntegrationTesting
	12.5	System Testing	
		12.5.R	Related_Patterns to SystemTesting
	12.6	Regression Testing	
		12.6.R	Related_Patterns to RegressionTesting
	12.7	Acceptance Testing	
		12.7.R	Related_Patterns to AcceptanceTesting
	12.8	Clean Room Testing	
		12.8.R	Related_Patterns to CleanRoomTesting

범주의 계층구조에서 SPO의 최고 깊이 수준(highest depth level)은 6.0, 평균 깊이 수준(average depth level)은 약 4.73으로 측정되었다.

4. SPO의 평가 및 특징

소프트웨어 패턴은 상향식 형태로 조직화됨으로써 패턴간의 엄밀한 추상성 정의에 어려움이 있다. 한편 온토로지는 특정 도메인을 대상으로 하여 도메인 내에서 사용하고 있는 어휘개념의 엄밀한 정의와 어휘개념

간의 계층분류를 목적으로 하고 있다. 온토로지는 이러한 목적을 성취하기 위해 특정 도메인내의 다중 에이전트들이 사용할 공통언어의 어휘개념을 하향식 방식으로 일반화하고 있다. 본 논문이 설계한 SPO는 소프트웨어 패턴의 기술(description) 프레임워크의 상향식 추상화(abstraction)와 온토로지의 하향식 일반화(generalization)를 통합하여 소프트웨어 패턴 지식정보의 체계를 형성한다.

소프트웨어 패턴은 소프트웨어 분석?설계 과정에서 산발적으로 도출된 개념이기 때문에 전체 도메인 관점에서 구조화가 미흡하다. SPO는 지식베이스의 기반을 이루는 온토로지 이론을 도입하여 소프트웨어 패턴내에 내재된 구조화된 상위지식 정보를 체계화하고 있다. SPO는 증가하는 소프트웨어 패턴에 대한 효율적인 분류와 검색을 지원할 수 있으며, 특히 새로운 유형의 소프트웨어 패턴 범주나 소프트웨어 패턴의 신규등록을 원할 때도 기존 시스템의 구조를 변경시키지 않고 등록 및 확장이 가능하다.

SPO에 적용된 소프트웨어 패턴 범주화는 소프트웨어 개발 생명주기의 과정을 기준으로 하였기 때문에 소프트웨어 개발과정에서 마주치는 임의의 문제점을 현실적 측면에서 구체적이고 체계적으로 해결할 수 있다. SPO에는 소프트웨어 패턴들 또는 객체간 관련성의 상태를 분류코드로 제시함으로서 소프트웨어 패턴 정보에 관한 이해의 효과를 높여준다.

V. 결 론

소프트웨어 패턴은 소프트웨어 개발과정에서 반복적으로 발생하는 문제에 대해서 통찰력 있는 전문가의 검증된 해결책을 제시해 주는 지식이다. 소프트웨어 패턴의 재사용은 소프트웨어 생산성을 제고시킴은 물론 객체지향 설계자들에게 객체지향 개념 습득의 어려움을 해소시키는 효과가 있다.

소프트웨어 패턴은 전문 소프트웨어 개발자의 경험과 지식의 표상이다. 따라서 본 논문에서는 전문 소프트웨어 개발자의 경험과 지식을 명확하게 명세화하고, 개발자간에 효율적인 커뮤니케이션 환경을 조성하여 소프트웨어 개발 지식을 공유하고 재사용하기 위한 공통의 언어로써 온토로지를 활용하였다. 온토로지는 지식 또는 개념 객체간의 연관성을 구조화하여 표현하는 것으로서 인공지능에서 지식 표현 방식의 토대가 되고 있다.

온토로지는 지식 개념의 명세화로서 체계적인 개념 분류를 토대로 하여 지식베이스의 골격이 된다. 소프트웨어 패턴은 소프트웨어 개발에서 파생되는 문제에 대한 해결책으로서 추상화하여 구조화한 것이다. 본 논문에서는 개념적으로 유사성이 있으나 적용 도메인이 상이한 소프트웨어 패턴과 온토로지를 융합하여 응용할 수 있는 방안을 보였다. 온토로지 개념을 이용하여 추상화된 소프트웨어 패턴의 개념을 조직적으로 분류하여 구조화하는 방법을 보였다. 따라서 소프트웨어 패턴 지식과 온토로지 개념의 결합으로 생성된 SPO는 소프트웨어 패턴 지식정보의 범주화 체제가 된다. 이는 소프트웨어 패턴에 관한 사용자의 정보검색 시스템과의 공유된 공통의 배경 지식을 명시하고 사용자의 요청과 정보제공자 사이에서 정보를 조정해 주며, 사용자 개체 범주를 할당하는 지식 에이전트 시스템 설계에 효과적인 상위의 메타정보체제의 기반이 된다.

본 논문에서 추출한 소프트웨어 패턴은 총 91개로, 150개의 개념이 등록되었으며, 객체 또는 개념과의 관계는 150건이 발생하였다. 그리고 이 같은 상황에서 SPO 구조 체계의 최고 깊이 수준은 6.0이었고, 평균 깊이 수준은 약 4.73으로 측정되었다.

향후 SPO의 개념화된 범주를 적용하여 소프트웨어 패턴을 문서로 형식화하고 패턴의 축적과 다양한 관리 기능을 갖는 패턴 응용 시스템이 구축되어야 할 것이다.

참 고 문 헌

[1] Booch, G., Object Oriented Design with Applications, Benjamin Cummings, 1993.

[2] Chandrasekaran, B., Josephson, J.R., and Benjamins, V.R., "What are ontologies, and why do we need them?", IEEE Intelligent Systems 14, 1, pp. 20~26, January-February 1999.

[3] Fowler, M., Analysis Patterns : Reusable Object Models, Addision-Wesley Publishing Company, Reading, MA, 1997.

[4] Gabriel, R., P., Patterns of Software : Tales From the Software Community, Oxford University Press, 1996.

[5] Gamma, E., Helm, R., Johnson, R., and

Vlissides, J., Design Patterns : Elements of Reusable Object-Oriented Software, Addison-Wesley Publishing Company, 1995.

[6] Grand, M., Patterns in Java volume 1 : A Catalog of Reusable Design Ptterns Illustrated with UML, John Wiley & Sons, Inc., 1998.

[7] Grand, M., Patterns in Java volume 2, John Wiley & Sons, Inc., 1999.

[8] Gruber, T.R., "A Translational Approach to Portable Ontologies", Knowledge Acquisition, Vol. 5, No. 2, pp. 199~220, 1993.

[9] Gruber, T.R., "The role of common ontology in achieving sharable, reusable knowledge bases", Allen et. al., eds. Principles of Knowledge Representation and Reasoning-Proceedings of the Second International Conference, pp. 601~602, 1991.

[10] Mattsson, M., "Object-Oriented methodological issues", Master Thesis, Development of Computer Science, University College of Karlskrona, 1996. 8.

[11] Menzies, T., "Cost Benefits of Ontologies", intelligence Fall 1999, pp. 26~32.

[12] O'Leary, D.E., "Different Firms, Different Ontologies, and No One Best Ontology", IEEE Intelligent Systems, September/October 2000, pp. 72~78.

[13] Pressman, R.S., Software Engineering : A practitioner's approach, McGraw-Hill, 1995.

[14] Riehle, D., and Zullighoven, H., "Understanding and Using Patterns in Software Development." Theory and Practice of Object Systems 2, 1, pp. 3~13, 1996.

[15] Schmidt, D.C., Fayad, M. and Johnson, R. E. "Software Patterns", Communications of the ACM 39, 10, pp. 37~39, 1996.

[16] SHOE(Simple HTML Ontology Extensions), Paralled Understanding Systems Group Department of Computer Science University of Maryland of College Park, (http://www.cs.umd.edu/projects/plus/SHOE/)

[17] Swartout, W. and Tate, A. "Ontologies", IEEE Intelligent Systems14, 1, pp. 18~19, January-February 1999.

[18] Zimmer, W., "Relationships Between Design Patterns", Pattern Languages of Programs 2, Vlissides et. al., eds., Addison Wesley Publishing Company, 1996.

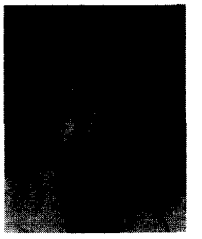
[19] 박경우, 김희수, 배상현, "Ontology를 이용한 지식 표현에 대한 연구", 조선대학교 통계연구소 논문집, 제99-1집, pp. 223~236, 1999.

[20] 양성기, "약 구조화 Ontology 기반 하에서 정보의 분류와 획득을 위한 지적 에이전트의 구현", 조선대학교 석사학위논문, 1998.

저 자 소 개

洪 顯 述(正會員)

1985년 : 원광대학교경영학과 졸업(경영학사). 1987년 : 원광대학교대학원 경영학과졸업(경영학석사). 1990년 : 원광대학교대학원 전자계산기공학과(공학석사). 2001년 : 원광대학교 대학원 컴퓨터공학과 졸업(공학박사). 1989년~현재 : 원광보건대학 컴퓨터응용개발과 교수. <주관심분야 : 객체지향시스템, 디자인패턴, WBI, XML, 온토로지시스템, 컴퓨터 응용>



韓 成 國(正會員)

1979년 : 인하대학교 전자공학과(공학사). 1981년 : 인하대학교대학원 전자공학과(공학석사). 1988년 : 인하대학교대학원 전자공학과(공학박사). 1984년~현재 : 원광대학교 전기전자 및 정보공학부 교수. 1988년~현재 : 미르칸전자 기술고문. 1989년~현재 : ㈜비트컴퓨터 기술고문. 1990. 12월~1992. 2월 : 미국 펜실버니아 대학교 Post-Doc. <주관심분야 : 인공지능(자연언어처리), 정보공학, 인지과학, 객체지향시스템, 컴파일러, XML, 컴퓨터교육, 온토로지시스템, 웹서비스>