패턴 정보를 이용한 설계패턴 검색 시스템 구축

김 귀 정† · 송 영 재††

요 약

본 연구는 설계패턴의 효율적인 관리와 재사용을 위하여 패턴 정보를 이용한 설계패턴 검색 시스템을 구축하였다. 패턴 정보는 패턴 속성정보와 패턴 메타정보로 구성하였고, 패턴 속성정보는 패턴 패싯 분류와 패턴 검색을 위한 유사도 측정에 이용되며, 패턴 구조를 UML로 모델링하기 위하여 패턴 메타 정보를 이용하였다. Gamma 분류 방법을 확장하여 각 설계패턴이 사용될 수 있는 여러 경험적 상황을 패싯 항목으로 설정하고 패턴 정보로 관리하였다. 또한 E-SARM 검색 방법을 사용하여 검색된 패턴은 메타정보를 이용하여 UML 클래스 다이어그램으로 나타낼 수 있도록 하였고, E-SARM을 설계패턴의 검색에 적용하여 최적의 결과를 얻을 수 있는 환경을 시뮬레이션 하였다. 패턴 뷰어를 통한 신규 패턴 등록이 가능하며, 등록된 패턴은 패턴 속성정보와 메타정보로 관리된다. 따라서 본 시스템은 효율적인 패턴 관리가 가능하고, UML 모델링을 지원하며, 관련 패턴의 우선순위 검색이 가능하여 패턴 선택 비용이 적고, 재사용성이 높은 설계패턴 검색 시스템이다.

키워드 : 설계패턴검색시스템, 설계패턴, 패턴정보, 패싯분류, 유사패턴, UML, E-SARM

Construction of Design Pattern Retrieval System using Pattern Information

Gui-Joung Kim† · Young-Jae Song††

ABSTRACT

In this paper, we implemented design pattern retrieval system for efficient management and reusability of design patterns. Pattern is consisted of property information and meta information. Property information is used for similarity measurement on classification and retrieval of patterns. Meta information is used for UML modeling of patterns. We classified design patterns with the empirical scope in addition to Gamma's basic classification. Also we used E-SARM for retrieval, represented UML diagram with pattern meta information, and simulated the environment so as to obtain best result on applying to retrieval of design pattern. This system is able to resister new patterns through pattern viewer and manages these patterns with property information and meta information. Thus this system supports efficient management of patterns, UML modeling, priority pattern retrieval, higher reusability and reduces pattern selection cost.

Key word : design pattern retrieval system, design pattern, pattern information, facets classification, relative pattern, UML, E-SARM

1. 서 론

객체지향 개발 방법론을 활용하기 위해서는 실제적인 개념과 규격화된 패턴, 상황에 따른 올바른 객체의 선택과 적용이 필요한데 이러한 객체지향 설계 개념을 이론적으로 제시하는 것으로 그치지 않고 프로그래밍에 적용 가능하도록 구체적으로 규격화시킨 것이 설계패턴이다[1]. 현재 PloP (Pattern Languages of Programs)와 EuroPLoP를 통해서 공식적으로 발표되어 알려져 있는 설계패턴은 수 백 가지에 이른다[2, 3]. 설계 단계에서 설계패턴을 고려해야 개발 도중 발생 가능한 시스템 변경에 보다 유동적으로 대처할 수 있기 때문에 설계패턴을 보다 쉽게 이해하고 적용하기 위해서는 UML(Unified Modeling Language)의 사용이 필수적이다[1]. 그러나 패턴의 체계적인 분류, 공유가 이루어지지 않아 적당한 패턴을 찾지 못하거나, 참조되지 못하는 경우가 많았다. 이는 효율적인 패턴 라이브러리(Design Pattern Library)를 구축함으로서 해결할 수 있으며, 패턴 정보 외에 패턴 구조를 컴포넌트화하여 제공하는 것이 필요하다[4]. 설계패턴에 대한 정보와 코드 생성을 제공하는 라이브러리 외에도 시스템 설계 시 설계패턴을 적용하는 CASE 도구들이 있다[5-7]. 이들 시스템들은 패턴 정보 외에도 시스템 설계 시 재사용 할 수 있는 패턴 구조를 제공하고 있지만, 독립적인 모델링의 도구 지원의 취약이나 비효율적인 패턴 분류와 검색방법, 신규 패턴 등록을 허용하지 않는 것과 같은 단점을 가지고 있다. OmniBuilder[5]는 애플리케이션 설계자들은 특정한 요구에 맞는 그들만의 설계패턴을 정의할 수 있으나, Gamma의 분류 방식을 따르며 특별한 검색 도구가 지원되지 않기 때문에 계속적으로 추가되는 패턴들을 효율적으로 관리하고 검색하는데 어려움

† 정 회 원 : 경희대학교 대학원 전자계산공학과
†† 종신회원 : 경희대학교 전자계산공학과 교수
논문접수 : 2000년 8월 28일, 심사완료 : 2001년 2월 2일

이 있다. 또한, ModelMaker[6]는 UML과 같은 객체 지향 방법론을 이용한 모델링 도구가 독립적으로 존재하지 않으며, IBM의 설계패턴 라이브러리[3]는 신규 패턴의 등록이 지원되지 못하여 Gamma의 기본 패턴만을 제공하는 한계를 갖는다.

이처럼 수 백 가지에 이르는 패턴을 효율적으로 재사용하기 위해서는 패턴이 수행하는 역할에 따른 3가지 분류와 패턴명 만으로 검색해야하는 Gamma의 분류 방법으로는 사용자가 원하는 패턴을 찾아 적용하는데는 많은 어려움이 따른다. 그러므로 상황에 맞는 적절한 경험을 자동화된 방식으로 제공할 수 있는 패턴 분류 방법과 검색, 그리고 새로운 패턴의 등록과 UML을 이용한 모델링 도구 지원 등의 제공이 요구된다. 따라서 본 연구는 Gamma 패턴 외에도 사용자에 의해 신규 패턴의 등록과 관리가 가능하며, 증가하는 패턴에 대한 효율적인 분류와 검색을 지원하고, 패턴이 UML로 표현되고 모델링 될 수 있는 설계패턴 검색 시스템을 구축하였다. 재사용성을 극대화하기 위해 패턴 구조를 재사용 컴포넌트로 사용될 수 있도록 구성하고, 패턴 구조를 가시적으로 모델링하기 위하여 UML 기반의 패턴 정보를 이용하였다. 패턴 정보는 일반적인 패턴의 속성정보를 가지고 있는 패턴 속성정보와 패턴 구조 정보를 가지고 있는 패턴 메타정보로 구성하였다. 패턴 속성정보는 컴포넌트화된 패턴의 재사용성을 향상시키기 위하여 Gamma의 패턴 분류 방법을 확장한 패싯 분류와 요구사항에 맞는 패턴을 검색하기 위한 유사도 측정에 이용된다. 또한 설계패턴의 구조를 UML로 모델링하고 컴포넌트화 시키기 위하여 설계패턴 구조로부터 추출한 클래스 메타모델, 속성 메타모델, 오퍼레이션 메타모델, 관계 메타모델 등과 같은 패턴 메타정보를 이용한다. 이러한 필요성에 부합한 시스템을 설계, 구현하기 위해서 MFC를 사용하였다. 패턴 등록을 비롯한 패턴 관리와 패턴구조 모델링, 검색의 사용자 인터페이스를 구현하고, 패턴 정보와 구조를 저장할 수 있는 데이터베이스를 구축하였다.

본 연구는 UML 기반의 패턴 정보를 이용하여 설계패턴 검색 시스템을 구축하였다. 또한 패싯 분류방법과 E-SARM의 유사 검색방법을 사용하여 Gamma의 분류방법으로는 부족했던 질의에 대한 정확성을 제공하였으며, 유사도를 이용해 관련 패턴까지 검색함으로써 효율적인 패턴 재사용이 가능한 시스템이다. 또한 검색된 패턴을 수정하고 새로운 패턴을 등록시킬 수 있으며 검색된 패턴을 UML 다이어그램으로 나타낼 수 있다. 본 시스템은 검색 인터페이스, 라이브러리 관리, 재사용 기술, 편집 기능, 다이어그램 설계 도구 기능으로 구성하였다.

2. 관련 연구

객체 지향 프로그램 개발을 위한 많은 도구들이 출현하였고, 설계를 위한 부품, 패턴 등을 지원함으로써 시스템의 효율적 구축을 목적으로 많은 연구가 진행되고 있다. OmniBuilder[5]는 설계 단계에서 패턴을 사용하는 도구로써 애플리케이션의 전체 생명주기를 거쳐 애플리케이션을 생성하는 CASE 도구이며, 복잡한 애플리케이션 개발에 필요한 시간과 비용을 감소시킨다. 설계패턴은 OmniBuilder안에서 보통의 객체처럼 모델링되고, 애트리뷰트·서비스·이벤트·메소드와 행위를 가질 수 있다. 또한 객체지향적인 언어로 개발자에 의해 프로그래밍될 수 있다. OmniBuilder는 복잡한 객체 행위를 구현하는 다양한 설계패턴을 제공하며, 애플리케이션 설계자들은 특정한 요구에 맞는 그들만의 설계패턴을 정의할 수 있다. 그러나 Gamma의 설계패턴과 그 분류 방식을 따르며 특별한 검색 도구가 지원되지 않기 때문에, 계속적으로 추가되는 패턴들을 효율적으로 관리하고 검색하는데 어려움이 있다.

ModelMaker[6]는 볼랜드 델파이를 위한 클래스와 컴포넌트 패키지를 개발하기 위한 CASE 도구이다. 클래스·유스케이스와 시퀀스 다이어그램을 지원하며, 클래스와 멤버들 사이에 모든 관계를 저장하고 유지하는 동적인 모델링 엔진을 가지고 있어 클래스명을 바꾸거나 상위클래스의 내용을 바꾸는 행위는 생성된 소스 코드에 즉시 반영된다. ModelMaker의 특징은 패턴이 관련된 코드에 패턴을 삽입한 후에 동적으로 유지된다는 것이다. ModelMaker에서 사용되는 패턴은 메소드를 오버라이드하는 ModelMaker의 능력을 사용해 쉽게 구현될 수 있어 패턴의 수를 감소시킬 수 있다. 그러나 UML과 같은 객체 지향 방법론을 이용한 모델링 도구가 독립적으로 존재하지 않고, 마치 매크로처럼 동작하는 템플릿만을 지원하기 때문에 패턴 구조를 다양하게 모델링하는데 어려움이 있다.

IBM의 설계패턴 라이브러리[3]는 설계패턴의 구현을 자동화하는 도구이다. 사용자가 주어진 패턴을 위한 특정 애플리케이션 정보를 제공하면, 이 도구는 자동적으로 규정된 패턴 코드를 생성한다. IBM의 DPL은 재사용 컴포넌트 구현에 적합한 분산 구조를 갖으며, 사용자 제공 정보로부터 자동적으로 설계패턴 코드를 생성한다. 또한 설계패턴에 온라인, 하이퍼텍스트 기능을 통합하였으며, 편리한 접근과 패턴들 사이에 즉각적인 링크와 빠른 정보 검색을 허용한다. 그러나 신규 패턴의 등록을 지원하지 못하여 Gamma의 기본 패턴만을 제공하는 한계를 갖는다. 또한 자체적인 모델링 도구나 방법을 지원하지 못한다.

3. 설계패턴 검색 시스템 설계

설계패턴 검색 시스템의 구조와 패턴 정보를 이용한 데이터베이스의 구축과 검색 시스템의 설계에 대해서 기술한다.

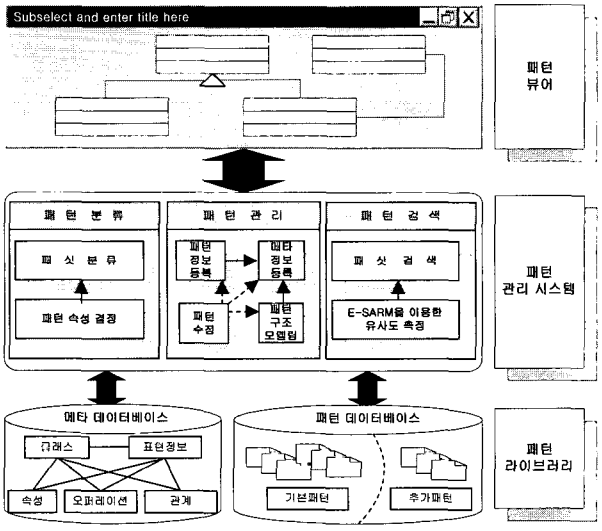
3.1 검색 시스템의 구조

기존의 시스템은 Gamma 패턴을 기본으로 하고 있으며, 일반적인 패턴 정보 외에 컴포넌트화된 패턴구조를 제공하고 있다. 그러나 사용자에 의한 신규 패턴의 등록이나 UML과 같은 모델링 도구의 지원이 충분치 못하거나, 패턴 관리와 검색이 효율적이지 못한 단점이 있다. 본 연구에서는 이러한 문제점들의 개선 사항을 반영한 설계패턴 검색 시스템을 설계·구현하였다.

3.1.1 시스템 구조

본 논문에서 제안한 설계패턴 검색시스템은 크게 패턴 라이브러리, 패턴 분류·관리·검색을 담당하는 패턴 관리 시스템, 그리고 패턴 뷰어로 구성되며, (그림 1)은 구현한 설계패턴 검색시스템의 구조를 보여준다.

패턴 라이브러리는 일반적인 패턴의 속성정보가 저장되는 패턴 데이터베이스와 UML로 표현된 패턴 구조를 메타모델로 저장하는 메타 데이터베이스로 구성된다. <표 1>은 2가지 패턴 정보를 나타낸 것이다.



(그림 1) 설계패턴 검색 시스템의 구조

〈표 1〉 패턴 정보

패턴 정보	데이터베이스	사용 범위
패턴 속성정보	패턴 데이터베이스	패싯 분류, 유사성을 이용한 검색
패턴 메타정보	메타 데이터베이스	UML 모델링

패턴 속성정보는 설계패턴을 분류하고 검색하는데 필요한 정보를 포함하고 있으며, 이 속성에 의해 패턴 데이터베이스에 저장된 패턴은 도메인 종속여부에 따라 기본 패턴과 추가 패턴으로 분류되어 저장된다. 패턴 메타정보는 패턴을 UML로 모델링하는데 필요한 정보를 포함하고 있다. 설계패턴 구조를 생성하기 위해서 패턴 등록 시 그 구조를 UML로 모델링하여 구성 정보를 데이터베이스화하였다.

패턴 관리시스템은 패턴의 분류와 등록, 검색, 패턴 정보 관리 등의 역할을 수행하여 라이브러리와 패턴 뷰어 사이의 데이터 흐름과 처리를 담당한다. 증가하는 패턴을 효율적으로 관리·검색하기 위해, 기존의 패턴명으로 검색하는 스트링 매칭 검색 외에 응용 도메인 종속 여부와 패턴이 속한 영역, 적용 목적 그리고 적용 범위에 의한 분류의 패싯 항목을 갖는 패싯 분류를 지원하며, 관련 패턴까지의 검색이 가능하도록 기존에 클래스 컴포넌트에 사용되었던 E-SARM을 패턴 검색에 적용하여 재사용성을 극대화시킬 수 있도록 하였다.

패턴 뷰어는 검색 결과 선택된 패턴을 클래스 다이어그램으로 보여주거나, 신규 패턴을 모델링하고 등록하기 위해서 사용된다. 또한 클래스의 속성을 편집하기 위한 편집기와, 검색 인터페이스, 패턴 추가·삭제의 패턴 관리기 등의 인터페이스를 제공한다. 편집기를 통해 모델링된 패턴 구조는 패턴 등록 시에 일반 패턴 속성정보와는 별도로 설계패턴을 구성하는 메타모델들을 이용하여 메타 데이터베이스를 구축하게 되고, 검색 시 호출되는 패턴은 메타 데이터베이스에 저장되어 있는 메타모델을 이용해 UML 모델링 규칙에 따라 클래스 다이어그램으로 표현된다.

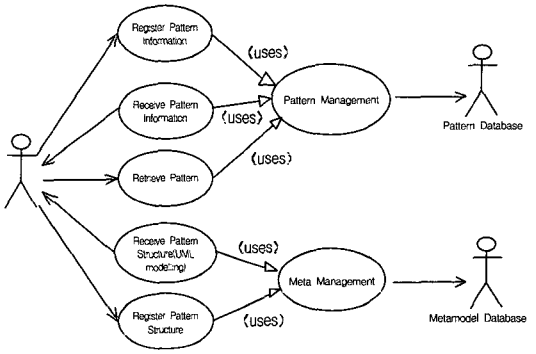
3.1.2 시스템의 UML 설계

유스케이스 다이어그램은 시스템과 사용자의 요구 분석 과정에 사용되는 다이어그램으로 시스템 사용자 혹은 외부 시스템을 나타내는 Actor와 Actor들과 관련하여 수행되는 행위인 Use Case의 상호 작용으로 표현된다. 본 시스템은 모두 3개의 Actor와 6개의 Use Case로 표현할 수 있으며, 이들에 대한 각각의 의미는 <표 2>와 같다. (그림 2)는 본 시스템의 유스케이스 다이어그램을 나타낸 화면이다. User 액터와 유스케이스들 사이에는 연관 관계가 존재하며, PatternManagement나 MetaModel Management 유스케이스는 다른 유스케이스들에게 공통으로 사용되는 유스케이스 역할을 한다. User 액터는 패턴을 등록, 검색하거나 패턴 정보를 얻을 수 있으며, 이러한 행위는 패턴 데이터베이스에 연결된 Pattern Management 유스케이스를 통하여 이루어진다. 일반적인 패턴 정보 외에도 User 액터는 패턴 구조를 MetaModel Management 유스케이스를 사용해 메타모델 데이터베이스에 등록하게 된다. (그림 3)은 추출된 공통적인 클래스와 객체 사이의 정적인 관계를 표현한 개략적인 클래스 다이어그램이다. Register Pattern Information 클래스와 Receive Pattern Information 클래스는 Pattern Information 클래스를 추가하여 일반화 관계를 설정하여 두 클래스들 간의 유사한 기능을 Pattern Information 클래스에 추상화하였다. 하나의 패턴 정보에는 하나의 패턴 구조가 있으므로 Pattern Information과 Pattern Structure 클래

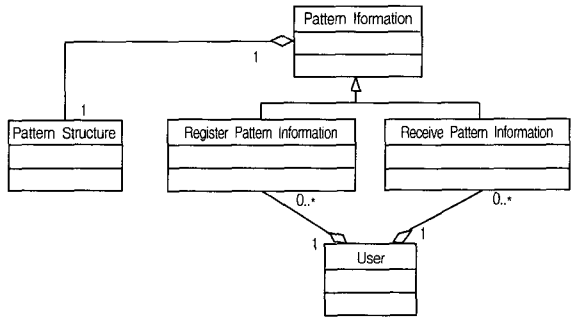
스 사이에는 일대일의 다중성이 성립된다. 하나의 User는 여러 패턴 정보를 등록하거나 전송 받을 수 있으므로, User 클래스와 Register Pattern Information 클래스와 Receive Pattern Information 클래스 사이에는 일대다의 다중성이 성립된다.

〈표 2〉 패턴검색 시스템의 Actor/Use Case

Actor/Use Case	의 미
User	사용자
Pattern Database	패턴정보를 저장하는 DB
MetaModel Database	메타모델정보를 저장하는 DB
Retrieve Pattern	패턴 검색
Receive Pattern Information	패턴정보 회수
Register Pattern Information	패턴정보 등록
Register Pattern Structure	패턴구조 등록
Pattern Management	패턴정보 관리
MetaModel Management	패턴메타모델 관리



(그림 2) 검색 시스템의 유스케이스 다이어그램



(그림 3) 검색 시스템의 클래스 다이어그램

3.2 설계패턴 정보

본 논문에서 구현한 설계패턴 검색시스템의 패턴 라이브러리와 패턴 뷰어에서의 모델링에 사용된 설계패턴 정보에 대해서 알아본다. 패턴 정보는 패턴 속성정보와 메타정보로 구분되고, 패턴 속성정보는 설계패턴을 분류하고 검색하는데 필요한 정보를 포함하고 있으며, 패턴 메타정보는 패턴을 UML로 모델링하는데 필요한 정보를 포함하고 있다.

3.2.1 패턴 속성정보

패턴의 속성정보는 기본적으로 Gamma의 설계패턴 표현에 사용되는 형식을 사용하며, 그 정보를 저장하는 패턴 데이터베이스에는 그 외에도 4가지 방법에 의한 패턴의 패싯 분류 정보와 재사용 정보가 추가로 저장된다. (그림 4)는 패턴의 속성정보에 사용된 데이터를 나타낸 것이다. 패턴 속성정보에는 패턴 이름, 패턴 파일명, 분류를 위한 패싯과 항목, 패턴구조의 이미지, Gamma의 설계패턴 표현 방법, 관련성, 패턴의 재사용 횟수, 구성 클래스의 수 등이 있다. 패싯과 항목들은 각 패턴이 패싯 항목 중 어느 부분에 속하는가를 나타내어 항목과 패턴 사이의 관계를 지정해 주는 역할을 함으로써 관련성을 표현할 수 있도록 하였다. 이 관련성은 검색 시 패턴간 유사성을 계산하는데 사용된다. 항목과 패턴의 연결 상태는 패턴이 분류된 패싯 항목에 따라 초기 값으로 설정되고, 이 항목과 패턴 사이의 연결을 통하여 활성값이 전달되어 항목과 관련된 패턴들을 검색할 수 있다. 재사용 횟수는 각 패턴이 어느 정도의 재사용성을 가지고 있는가를 보여주어 재사용 정도를 평가하고 재사용 패턴들에 대한 효율적인 관리가 가능하도록 해준다. 설계패턴들은 특정 응용 도메인에 상관없이 적용될 수 있는 도메인 독립적인 기본 패턴과 특정 응용 도메인에 따라 선택적으로 적용될 수 있는 도메인 종속적인 추가 패턴으로 구분된다. 도메인 독립적인 기본 패턴은 Gamma가 정의한 23개 패턴을 이용하였으며 도메인 종속적인 추가 패턴은 사용자가 특정 응용에 따라 추가로 저장한 패턴을 포함한다.

패턴 정보 : 테이블

필드 이름	데이터 형식	
index	일련 번호	일련번호
pattern	텍스트	패턴이름
pattern_name	텍스트	저장된 패턴 화일명
strDomain	텍스트	도메인 종속 여부
strScope	텍스트	영역에 따른 분류
strPurpose	텍스트	사용 목적에 따른 분류
strEmprical	텍스트	실제 응용 상황
image	OLE 개체	패턴 구조 이미지
Intent	텍스트	원리와 의도
Motivation	텍스트	패턴의 추상적 기술의 이해를 돕는 예제
Participant	텍스트	디자인패턴에 참여하는 클래스와 오브젝트들
Consequence	텍스트	패턴 사용의 결과
Application	텍스트	패턴이 적용될 수 있는 상황
reusecount	숫자	재사용횟수
reuserate	숫자	패턴의 재사용률
class 갯수	숫자	구성 클래스 갯수

(그림 4) 패턴속성 정보 테이블

3.2.2 패턴 메타정보

설계패턴을 UML로 모델링하고 컴포넌트화 시키기 위하여 패턴 구조를 메타 데이터로 저장하여 패턴 메타 데이터베이스로 구축하였다.

<표 3>은 클래스와 인터페이스 메타모델의 정보를 갖는 테이블 구조이며, 인터페이스는 스테레오타입이 Interface인 클래스로 기술된다. Visibility 필드는 접근권을 말하며, UML에서는 특정 언어에 독립적인 방법으로 표기한다. Stereotype 필드는 클래스를 한 단계 높은 차원에서 분류한 것이고, Constraint 필드는 제약 사항을 말하며 정형화된 규칙은 존재하지 않는다[8].

〈표 3〉 클래스/인터페이스 메타모델

필드명	데이터 형식	의 미
ID	varchar	클래스, 인터페이스의 ID
Link_Pattern	varchar	소속된 설계패턴명
Name	varchar	클래스, 인터페이스명
Visibility	varchar	가시성
Stereotype	varchar	스테레오타입
Constraint	varchar	제약 조건

<표 4>는 <표 3>의 클래스와 인터페이스에 속한 애트리뷰트와 오퍼레이션 메타모델을 정보로 갖는 테이블의 구조이다. 이중 Value 필드는 애트리뷰트에만 해당하는 메타데이터이다.

<표 5>는 Dependency, Associations, Generalization, Refinement, Aggregation 같은 관계 메타모델들의 테이블 구조이며, 관계가 시작되고 끝나는 메타모델들의 ID를 기술한다. <표 6>은 패턴을 구성하는 모든 메타모델들의 표현정보를 저장하는 테이블 구조를 나타내며, 모델이 그려지는 좌측상단의 X, Y좌표, 폭, 높이와 같은 좌표 정보와 폰트, 선 색깔과 같은 스타일 정보를 기술한다. 이 정보를 이용해서 패턴 구조의 시각적 표현이 이루어진다.

〈표 4〉 애트리뷰트/오퍼레이션 메타모델

필드명	데이터 형식	의 미
ID	varchar	애트리뷰트/오퍼레이션 ID
Link_Class	varchar	소속된 클래스/인터페이스명
Name	varchar	애트리뷰트명/오퍼레이션명
Type	varchar	데이터 타입
Value	varchar	애트리뷰트값
Visibility	varchar	가시성
Stereotype	varchar	스테레오타입
Constraint	varchar	제약 조건

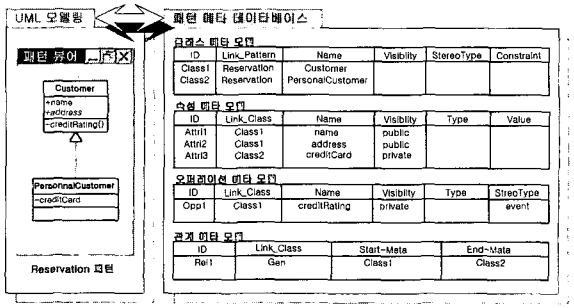
〈표 5〉 관계 메타모델

필드명	데이터 형식	의 미
ID	varchar	관계 ID
Name	varchar	관계명
Start_Meta	varchar	관계가 시작되는 메타모델
End_Meta	varchar	관계가 끝나는 메타모델

〈표 6〉 메타모델 표현정보

필드명	데이터 형식	의 미
ID	varchar	표현하고자 하는 메타모델 ID
Geometry	Integer	좌표
Style	Integer	스타일(Font, LineColor, FillColor 정보)

패턴 뷰어에서 UML로 모델링된 패턴 구조는 (그림 5)와 같이 각 메타 모델들을 추출해 메타모델 정보를 갖는 메타데이터베이스에 저장된다. 패턴명이 Reservation인 패턴에서 Customer 클래스와 Personal Customer 클래스, 각각의 애트리뷰트와 오퍼레이션, 그리고 관계를 추출하여 메타모델 데이터베이스를 구축하였다. UML을 기반으로 한 패턴 구조 모델링을 지원하기 위하여 패턴 구조를 구성하는 메타모델을 이용하였다. 패턴 등록 시에 패턴 모델링 모듈이 수행되며, 사용자가 설계패턴 구조를 모델링하면 일반 패턴의 속성정보는 패턴 속성정보 데이터베이스에, 패턴 구조는 메타 데이터베이스에 저장된다.



(그림 5) 패턴 메타정보의 데이터베이스 구축

3.3 패턴 뷰어

패턴 뷰어는 검색 결과 선택된 패턴을 적용시키거나 새로운 패턴을 생성하기 위한 인터페이스로 제공된다. 패턴뷰어는 CASE 도구의 클래스 다이어그램과 유사하며, 제공되는 메타모델로는 클래스, 관계, 노트, 텍스트가 있다. UML ver 1.1의 클래스 다이어그램 표기법에 따라 패턴 구조를 표현하였으며, 패턴 적용 후 클래스 이름과 애트리뷰트, 오퍼레이션과 같은 클래스의 속성은 편집기를 사용하여 편집할 수 있도록 하였다. 편집기를 통해 모델링된 패턴 구조는 패턴 등록 시에 일반 패턴 속성정보와는 별도로 설계패턴

```
Read Class_Size(rectangle, position), stereoType,
        Constraint from Data_File
Read relationship_ID from Data_File
For Link_Node Count
    Read Link_Node_ID
    Read Relation_Info related Classes
    Read Attribute_Count and Method_Count
    For Attribute_Count
        Create Attribute_Node
        Read Attribute_Info from Data_File
        Append Linked_List
    For Method_Count
        Create Method_Node
        Read Method_Info from Data_File
        Append Linked_List
For Class_Count
    Draw each Classes
Read Line_ID
For each Object_Count
     each Relationship assigned one of dependancy,
            realization, association,
            generalization, and aggregation
For each Object_Count
    Choose line_type (dependancy, realization,
            association, inheritance, aggregation)
    Draw lines between Objects
```

(알고리즘 1) 클래스 다이어그램 표현 알고리즘

메타모델들을 통하여 메타 데이터베이스를 구축하게 된다.

클래스다이어그램 뷰어 알고리즘은 패턴 메타 데이터베이스에 저장되어 있는 패턴 데이터 파일에서 각 클래스, 애트리뷰트, 오퍼레이션, 관계정보 등의 메타모델을 이용하여 클래스 다이어그램으로 표현해 준다. 뷰어에서 표현된 클래스나 라인을 직접 이동시키면 자동으로 클래스와 관계정보가 이동된다. 이는 Bresenham's Line algorithm[11]을 이용하여 구현하였다.

4. 설계패턴 검색 시스템 구현

패턴을 속성에 따라 분류하여 패턴을 쉽게 검색할 수 있도록 패턴이 수행하는 기본적인 역할과 영역, 적용 범위 등에 관한 정보를 패싯 항목으로 선택할 수 있도록 하였다. 또한 관련 패턴의 검색을 지원하여 다수의 관련 패턴을 제공함으로써 사용자 선택의 폭을 넓히고 효율적인 재사용이 가능하도록 하였다. 패턴 등록 및 삭제, 수정이 가능한 인터페이스를 지원하며 검색된 설계패턴을 UML 클래스 다이어그램으로 표현할 수 있도록 패턴 구조에서 추출한 패턴 메타 데이터를 이용하였다.

4.1 설계패턴 분류

설계패턴은 기존의 컴포넌트와는 달리 클래스, 애트리뷰트·오퍼레이션 이름이 각 패턴의 역할을 나타내기 위한 임시 이름일 뿐 실제 설계 시 적당한 이름으로 다시 붙여지게 되며 내부 작동 코드도 거의 존재하지 않는다. 이와 같이 설계패턴은 패턴을 구성하고 있는 내부 요소에 대한 정보보다는 패턴이 어느 상황에 적용될 수 있는가에 대한 경험적 정보를 가지고 있는 것이 재사용을 효율적으로 수행하는데 도움이 될 것이다. 따라서 본 연구에서는 각 설계패턴이 사용될 수 있는 여러 경험적 상황을 패턴 정보로 관리하고 이 정보를 이용하여 패싯 항목으로 설정하는 패싯 분류방법을 사용하였다.

패싯은 패턴의 기본적인 기능과 목적 그리고 패턴을 구현하고 실행할 수 있는 실제 상황을 기술할 수 있도록 도메인 종속여부, 패턴이 속한 영역, 적용 목적, 적용 범위의 4개 패싯을 포함하도록 정의하였다. 각 패싯의 특성을 표현하는 항목을 <표 7>과 같이 정의하였고, 이 항목에 해당하는 대표적인 Gamma의 설계패턴을 나타내었다. 4개의 패싯과 13개의 항목은 패턴 속성정보로 저장되어 각 패턴이 어떤 항목에 속해있는가에 따라 패싯 분류가 이루어지고, 이를 바탕으로 검색이 이루어진다. 도메인 종속 여부는 각 패턴의 재사용 시 특정 응용 도메인에 국한된 패턴인지를 선택하는 기준이 된다. 또한 사용자가 특정 도메인에만 적용 가능한 패턴을 따로 관리, 검색할 수 있도록 하였다. 영역은 각 패턴의 적용에 대한 범위로서 클래스와 객체로 구분되며, 패턴이 하는 역할인 목적에 따라 생성 패턴, 구조 패턴, 행위 패턴으로 구분된다. 마지막으로 적용 범위에 따라 재사용 가능한 상황에 초점을 두어 6가지 경험적 솔루션을 제시하였다. 각 패턴은 하나 이상의 적용 범위에 대한 속성을 가질 수 있어 실제 설계패턴이 재사용될 수 있는 여러 상황을 제시하여 경험을 바탕으로 재사용 패턴을 선택함으로써 재사용성을 최대화할 수 있도록 하였다. 또한 새로운 패턴을 등록할 때 각 패싯에서 패턴의 특징에 맞는 항목을 선택하고 특히 적용 범위에 대해서는 하나 이상의 항목 선택이 가능하도록 하였다.

적용 범위에 대한 패싯 항목 중 첫 번째는 시스템 요구사항을 객체로 분리시키는 솔루션으로 객체지향 설계의 내용이 실세계와 연관이 적은 클래스로 구성되는 경우나 추상화하기 힘든 개념을 프로그래밍에서 다룰 수 있는 객체로 변환시키는 경우에 사용할 수 있도록 하였다. 두 번째는 시스템 기능을 객체로 분할하기 위한 솔루션으로 큰 기능의 시스템 모듈을 객체 단위의 서브 모듈로 표현할 때 또는 많은 모듈을 효율적으로 관리하고자 할 때 사용할 수

<표 7> 각 패싯의 항목과 설계패턴

패 싯		특 성
도메인 종속 여부		도메인 독립(Factory Method외 Gamma의 22개 패턴) 도메인 종속(사용자 정의 추가 패턴)
Gamma 분류	영역	클래스(Factory Method, Adapter, Interpreter, Template Method) 객체(Abstract Factory, Builder, Prototype, Singleton, Bridge, Composite, Proxy, Decorator, Facade, Flyweight, Chain of Responsibility, Command, Iterator, Mediator, Memento, Observer, State,Strategy, Visitor)
	목적	생성패턴(Factory Method, Abstract Factory, Builder, Prototype, Singleton) 구조패턴(Adapter, Bridge, Composite, Proxy, Decorator, Facade, Flyweight) 행위패턴(Interpreter, Template Method, Chain of Responsibility, Command, Iterator, Mediator, Memento, Observer, State, Strategy, Visitor)
적용범위		요구 사항을 객체로 분할(Composite, Strategy, State) 시스템 기능을 객체로 분할(Abstract Factory, Builder, Facade, Flyweight, Visitor, Command) 객체의 인터페이스 지정(Proxy, Decorator, Memento, Visitor) 객체 지향 프로그래밍 적용(Factory Method, Abstract Factory, Builder, Prototype, Singleton, Composite, Chain of Responsibility, Command, Observer, State, Strategy) 델리게이션 메카니즘 적용(Bridge, Chain of Responsibility, State, Strategy, Visitor, Mediator) 런타임, 컴파일타임 구조의 관계규정(Composite, Decorator, Observer, Chain of Responsibility)

있도록 항목으로 제시하였다. 세 번째는 객체의 인터페이스를 지정하는 솔루션으로 인터페이스의 종류와 인터페이스 간의 연관 관계를 지정하는 경우에 사용할 수 있도록 하였으며, 네 번째 항목인 실제 객체 지향 프로그래밍에 적용 가능한 솔루션은 실제 프로그램을 작성 할 때 클래스 상속과 인터페이스 상속의 차이에서 오는 문제를 해결하거나 객체 생성 시 객체 연동에 필요한 인터페이스의 구현에 해결책을 줄 수 있도록 하였다. 또한 실제 구현 시 개발자가 코드를 재사용할 때 상속 기법의 재사용성이 포함된 복합(Composition) 기법인 델리게이션(Delegation) 메카니즘을 사용함으로써 재사용성이 증가할 수 있도록 하였다. 마지막으로 런타임 구조·컴파일 타임 구조의 관계를 규정하는 솔루션은 복잡한 런타임 구조를 구축하거나 런타임에 대한 구조와 컴파일 타임 구조와의 구분을 명확히 하기 위하여 사용할 수 있도록 하였다[10].

4.2 패턴 검색 모델

본 연구에서는 설계패턴의 각 속성을 패싯 항목으로 설정하여 패턴을 분류하였는데 이러한 분류를 기본으로 E-SARM[9]을 이용하여 패턴을 검색하였다. 이 방법은 사용자가 원하는 것을 분명히 표현하지 못했을 때도 사용자가 원하는 결과를 최대한 생성한다. 선택한 각 항목은 패턴의 속성을 나타내는 질의어의 역할을 수행하므로 하나 이상의 복수 개 질의어에 의한 검색이 이루어지며 선택된 항목에 대해 직접 연관된 패턴뿐 아니라 간접적으로 영향을 받는 패턴까지도 검색이 가능하다. 이는 패싯 분류로 인하여 발생하는 패턴 간 연관성 표현의 어려움을 해결해주는 역할을 한다. 그러나 기존의 E-SARM은 클래스 부품에 적용한 검색 방법으므로, 이를 패턴 검색에 적용하기 위하여 본 연구에서는 유사도 측정 방법과 참조 횟수에 의한 제거 기준의 선택 방법을 재설정하였다. 유사도는 각 패싯 항목별 검색 결과의 평균치로 설정하였고, 제거 기준을 결정하기 위하여 참조 회수를 이용하여 시뮬레이션하였다.

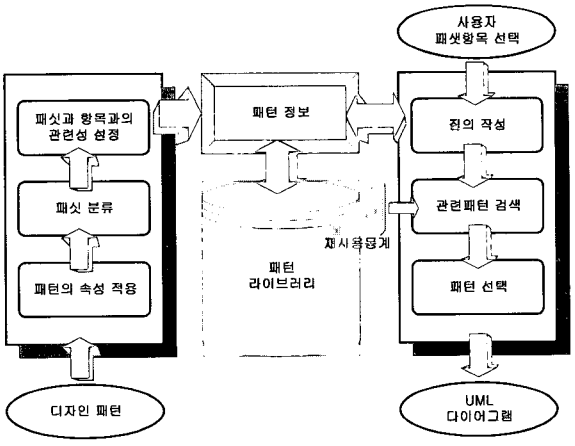
선택할 수 있는 패싯 항목이 4개이므로 기존의 E- SARM 방법으로 유사도를 측정할 수 없다. 그러므로 각 선택 항목에 따라 최대 4번까지의 E-SARM 방법을 수행하여 각 검색 결과가 관련 패턴의 유사도 순으로 나타나도록 한다. 그후 각 항목에 대해 추출된 관련 패턴들을 비교하여 모든 항목에 대해 공통적으로 검색된 패턴들만을 선택해 유사도의 평균을 구하여 최종 검색결과로 사용하였다.

검색 시간을 최소한으로 줄이고 결과는 최대가 되기 위한 제거 기준을 결정하기 위해 참조 회수를 이용하여 시뮬레이션 하였다. 우선 평균 참조 회수를 가장 큰 기준으로 설정하고, 각 순환마다 노드간 연결의 평균 참조 회수의 10%, 50%, 80%를 각각 기준으로 설정하여 검색결과와 활성값 계산 회수를 산출하였다. 시뮬레이션 결과 활성값 계산 회수는 최소한으로 하면서 검색 결과는 최대가 되는 82%를 제거 기준으로 설정하였다. 시뮬레이션 결과는 5장 성능 평가 부분에 나타나 있다.

4.3 설계패턴 검색

패턴 검색 과정은 (그림 6)과 같다. 특성에 따라 패싯 분류된 패턴은 적용 도메인 영역에 따라 분리되어 관리된다. 사용자의 요구에 따라 각 패싯 항목이 선택되는데 다양한 재사용 상황과 적용 영역, 역할 등의 항목을 제공하여 적합한 패턴을 검색할 수 있도록 해주고, 유사도에 의해 다수의 재사용 가능한 관련 패턴까지 검색해줌으로써 패턴 선택의 폭을 넓힐 수 있도록 하였다. 선택된 패턴은 UML 클래스 다이어그램으로 표현되어 재사용된다.

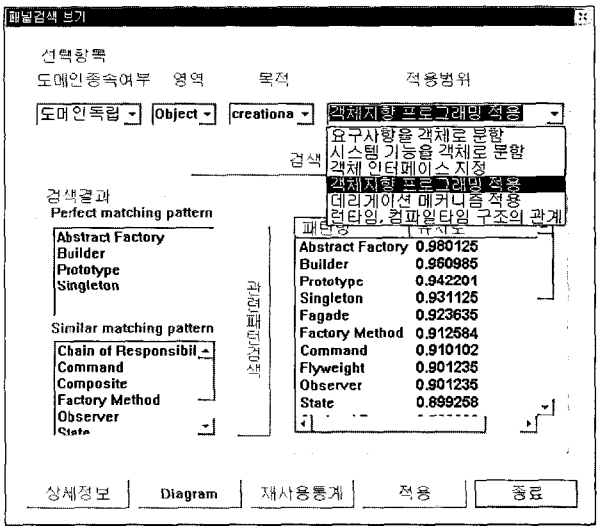
라이브러리에 저장되어 있는 패턴은 특성별로 패싯으로 분류되고 검색 시 각 패싯 항목을 선택함으로써 패싯 항목과 연결된 패턴들 사이의 활성값이 계산된다. 계산 도중 참조 회수가 적은 연결은 끊어지게 되고 최종적으로 계산된 활성값은 비슷한 유사도를 가지는 관련 패턴으로 검색된다. 검색된 다수의 관련 패턴들에 대해서는 패턴 적용의 도메인 종속 여부, 영역, 목적, 적용 범위 등의 패싯 항목에 대한 속성과 구성 클래스 수, 패턴 파일명 등의 상세 정보가 제공되며 각 패턴의 재사용 횟수를 그래프로 보여줌으로써 전반적인 패턴의 재사용성을 평가할 수 있어 효율적인 재사용이 가능하도록 하였다.



(그림 6) 설계패턴 검색 과정

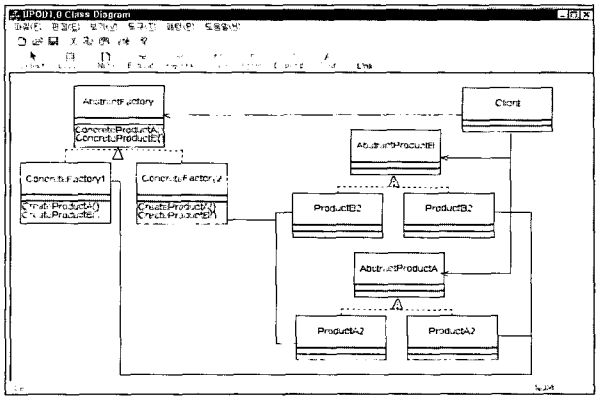
(그림 7)은 패싯에 대한 각 항목을 선택하여 검색한 결과를 보여준다. 검색결과에서 「Perfect matching pattern」은 선택한 4개의 항목을 모두 만족하는 패턴 리스트를 나타내며 「Similar matching pattern」은 4개의 항목 중 3개의 항목만을 만족하는 패턴 리스트를 나타낸다. 「관련패턴검색」기능은 선택한 항목에 따라 계산된 활성값에 의해 유사도가 오른쪽 창에 리스트로 제공될 수 있도록 해주며 재사용될 패턴을 선택하기 전에 각 패턴의 자세한 정보와 다이어그램의 형태를 보기 위해서 「상세정보」와 「Diagram」 기능을 사

용할 수 있다. (그림 7)은 "도메인 독립", "object", "creational", "객체지향프로그래밍 적용"의 패싯 항목을 각각 선택하여 검색한 결과를 나타낸다. 검색 결과 참조 횟수가 가장 많은 Abstract Factory 패턴을 중심으로 11개의 패턴이 검색되었는데 선택한 항목들과 직접 연결된 패턴은 상위 4개인 Singleton 패턴까지이고 나머지는 간접 연결된 관련 패턴들이다.

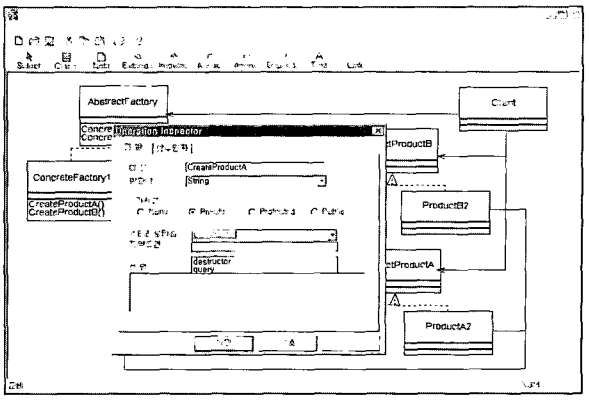


(그림 7) 패턴 검색

(그림 8)은 검색된 관련 패턴 리스트 중 Abstract Factory패턴을 선택했을 때의 클래스 다이어그램을 보여준다. 각 클래스의 이름과 애트리뷰트, 오퍼레이션과 같은 속성은 클래스 박스를 더블 클릭함으로써 편집 가능하며, (그림 9)는 패턴 적용 후 클래스에 대한 속성을 편집하기 위하여 속성을 입력하는 편집기 인터페이스를 나타낸다. Abstract Factory패턴을 모델링한 화면으로 참여자들 중의 하나인 ConcreteFactory1 객체의 속성 다이얼로그 창을 통하여 일반 정보와 애트리뷰트, 함수의 내역을 입력한다. 입력된 내용은 메타 데이타베이스에 저장된다.



(그림 8) Abstract Factory 패턴의 적용 예

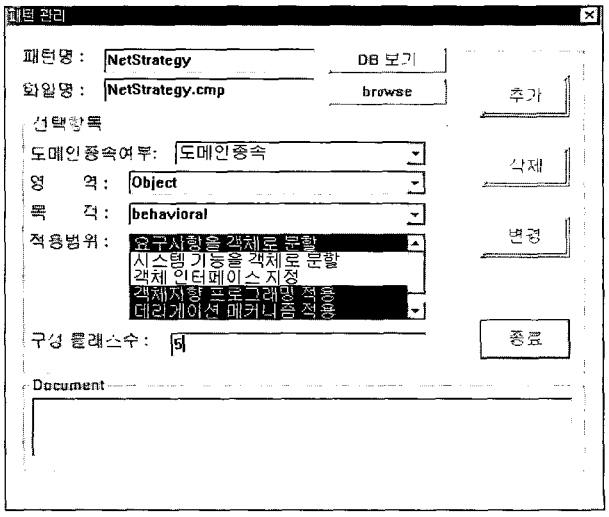


(그림 9) 클래스 속성 편집

4.4 패턴 관리

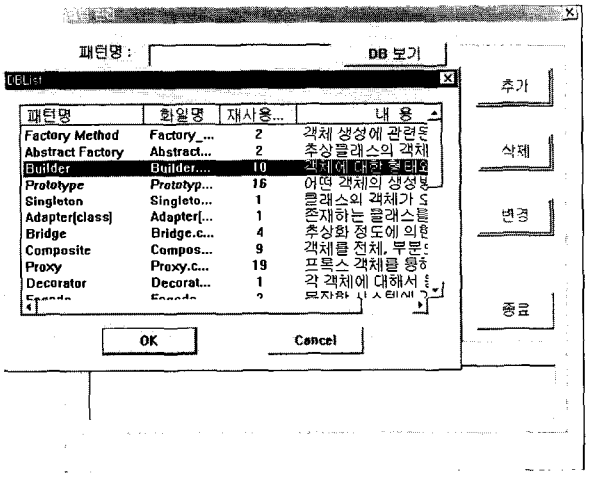
본 연구에서는 검색할 패턴의 확장성을 고려하여 현재 라이브러리에 있는 패턴들에 추가로 삽입하여 검색이 가능하도록 하였으며, 추가된 패턴과 항목들과의 연결관계가 자동적으로 패턴 정보로 저장되도록 하였다. 또한 삭제나 수정 시 라이브러리에 있는 패턴의 데이터베이스 정보를 제공하여 삭제·수정하고자 하는 패턴에 대한 정보를 미리 알 수 있도록 하였다.

(그림 10)은 "NetStrategy"라는 새로운 패턴을 추가하는 과정을 보여준다. 추가할 새로운 패턴은 클래스와 관계 그리고 텍스트와 같은 속성을 UML의 클래스 다이어그램으로 표현한 다음 인터페이스를 이용하여 추가할 수 있도록 하였다. 패싯 항목 중 적용 범위에 대해서는 패턴의 실제 적용 상황에 따라 여러 개 항목의 선택이 가능해야 하므로 각 항목들이 다중 선택 가능하도록 하였다. 추가된 패턴은 라이브러리에 저장되고 선택된 항목에 따라 항목과 패턴과의 관련성이 설정되며, 패턴 속성정보로 관리되기 위해 패턴 데이터베이스에 저장된다. 패턴 삭제와 변경 시에는 현



(그림 10) 패턴 추가

재 라이브러리에 있는 패턴의 정보를 알고 있어야 처리가 가능하므로 라이브러리의 패턴 정보를 데이터베이스의 테이블 형식으로 제공하였다. (그림 11)은 패턴 삭제와 변경 시 패턴의 정보를 이해할 수 있도록 테이블 형식으로 표현된 각 패턴의 속성을 보여준다.



(그림 11) 패턴 데이터베이스

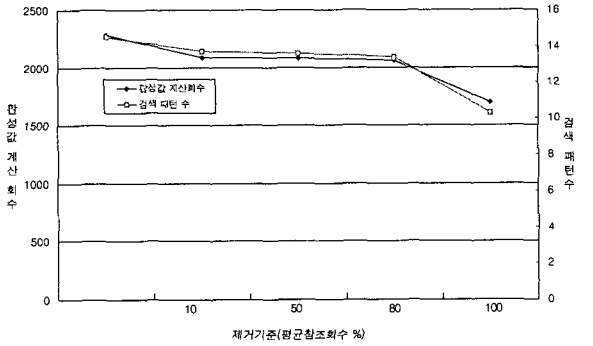
5. 성능 평가

본 연구는 설계패턴의 재사용성을 최대화하고 사용자의 요구사항과 일치하는 패턴이나 관련 패턴들을 효율적으로 검색하기 위하여 패턴 속성정보를 관리하였으며, UML의 클래스 다이어그램 패턴 구조로부터 패턴 메타모델 정보를 추출하여 라이브러리로 관리함으로써 효율적인 설계패턴 검색 시스템을 구축할 수 있었다.

기존의 E-SARM[9]은 클래스 부품에 적용한 검색 방법이므로, 이를 패턴 검색에 적용하기 위하여 본 연구에서는 참조 횟수에 의한 제거 기준을 재설정하였다. 검색 시간을 최소한으로 줄이기 위한 제거 기준을 결정하기 위해 참조 회수를 이용하여 시뮬레이션 하였다. 시뮬레이션을 위한 항목과 패턴은 13×23 크기의 배열을 이용해 구성하였다. 배열의 행은 항목을, 열은 패턴을 나타내고, 각 열과 행의 초기 값은 항목과 패턴의 연결 상태를 나타낸다. 이 항목과 패턴 사이의 연결을 통해 활성값이 전달되고, 항목과 관련된 패턴들을 검색할 수 있도록 하였다. 우선 평균 참조 회수를 가장 큰 기준으로 설정하고, 각 순환마다 노드간 연결의 평균 참조 회수의 10%, 50%, 80%를 각각 기준으로 설정하여 검색결과와 활성값 계산 회수를 산출하였다. 제거 기준을 정하지 않고 안정된 상태가 될 때까지 계속해서 실행했을 경우와 비교하여 검색 결과의 차가 가장 작으면서, 계산 회수가 적은 제거 기준을 찾았다. <표 8>은 각각의 경우에 대한 활성값 계산 회수와 검색된 패턴 수를 비교한 것이다. 이는 (그림 12)와 같이 제거 기준이 커짐에 따라 활성값 계산 회수는 줄어들고, 검색 결과 또한 줄어든다는 것을 보여준다. 이를 바탕으로 활성값 계산 회수는 최소한으로 하면서 검색 결과는 최대가 되도록 하기 위해 두 그래프의 교점을 제거 기준으로 설정하였다. 그래프에서 나타나는 두 번째 교점인 82%를 제거 기준으로 설정하였다.

〈표 8〉 제거 기준에 따른 활성값 비교

질의 (36)	연결을 제거시키지 않은 경우	연결을 제거시킬 경우			
		평균참조 회수의 10%	평균참조 회수의 50%	평균참조 회수의 80%	평균참조 회수
활성값 계산회수	2289.56	2087.95	2078.45	2063.43	1698.23
검색 패턴수	14.5	13.7	13.6	13.4	10.3



(그림 12) 제거기준에 따른 비교

본 시스템의 패턴 재사용 만족도를 <표 9>에서 보여주고 있는데, 이 실험은 임의의 패턴 검색 질의에 대하여 검색된 결과 패턴이 직접 연결된 패턴인지, 간접 연결된 패턴인지를 구별하여 비율로 산출한 것이다. 이 실험은 E-SARM 알고리즘을 패턴 검색에 적용함으로써 관련 패턴까지 검색되어 직접 연결 검색보다 선택의 폭이 14% 더 넓다는 결과를 보여주고 있다. 또한 <표 9>에서는 패턴의 재사용성을 보여주고 있다. 이는 검색된 패턴이 유사도 순으로 검색되기 때문에, 연결강도가 큰 패턴부터 선택하여 원하는 패턴인지를 판단하여 원하는 패턴이 선택된 확률을 구한 것이다. 이 결과는 활성값이 높을수록 패턴이 선택될 확률이 높다는 것을 의미하며 재사용성이 그 만큼 높다는 것을 나타내는 결과이다.

〈표 9〉 패턴 재사용 만족도

질의횟수 (36)	전체 검색된 패턴수		직접 연결 패턴수		관련 패턴수		직접 연결 패턴 선택비율		관련 패턴 선택비율	
평균	6.69		0.86		5.78		0.82 (평균 비율)		0.14 (평균 비율)	
활성값순	1	2	3	4	5	6	7	8	9	10
평 균 재사용성	37.5	25	25	5	2.5	2.9	2.0	0	0	0.1

그리고 평균 재사용성을 기준으로 패턴을 선택할 수 있는 비용(Cost)면에서 직접 연결에 의한 검색과 관련 패턴의 검색을 비교해 보면, 먼저 직접 연결 검색에서는 하나씩 패턴을 검사해야 하기 때문에, 직접 연결 검색에 의한 총 컴포넌트 검색비용(V_{PC})은

$$V_{PC}=C\cdot\frac{1}{10}+2C\cdot\frac{1}{10}+\cdots 10C\cdot\frac{1}{10}=5.5C \qquad (1)$$

이다. 여기서 비용은 패턴마다 모두 같다고 가정하여 C로 정의한다. 다음으로 관련 패턴 검색에 의한 전체 패턴 검색 비용(V_{SC})은

$$V_{SC}=CP_1+2CP_2+3CP_3\cdots+10CP_{10}=2.274C \qquad (2)$$

이다. 여기에서 $P_1, P_2, \cdots P_{10}$ 은 <표 9>의 평균 재사용률을 나타낸다. 이 결과는 관련 패턴 검색이 직접 연결 검색 방법보다 패턴 선택 비용이 적다는 것을 보여준다. 그 이유는 유사도 순으로 검색되기 때문에 직접 연결 검색보다 재사용 할 패턴을 보다 더 쉽게 찾을 수 있다는 의미이다. 따라서 본 연구에서 사용한 패턴 관련성을 이용한 검색 방법이 직접 연결 검색 방법보다 패턴 선택 비용이 절감됨을 알 수 있었고, 활성값 순으로 검색되기 때문에 보다 빨리 원하는 재사용 패턴을 검색할 수 있다는 결과를 증명할 수 있었다.

<표 10>은 검색 모델, 분류 모델, 유사패턴 검색, 모델링 도구지원 등의 특성을 기준으로 본 연구에서 설계한 설계패턴 검색 시스템과 기존의 시스템과 비교하였다. CASE 도구로 많이 사용되고 있는 Plastic이나 MetaEdit+는 UML 모델링은 가능하지만 독립된 검색 모델이나 분류모델이 존재하지 않기 때문에 컴포넌트의 관리와 검색이 효율적으로 이루어지지 못한다. 또한 패턴 기반 시스템인 Omibuilder와 ModelMaker는 Gamma의 패턴 분류를 기본으로 하고 있으며, 단지 패턴명만으로 검색 가능한 스트링 매칭 방법을 사용하기 때문에 계속적으로 추가되는 패턴을 효율적으로 관리하고 검색하는데 어려움이 있다.

〈표 10〉 기존의 시스템과 비교

항 목 / 시스템	컴포넌트	검색 모델	분류 모델	패턴/컴포넌트추가	확장검색(유사패턴)	모델링 도구 지원
OOCRS [9]	클래스	E-SARM	열거+패싯	○	○	컴포넌트(클래스)
Plastic [12]	클래스	스트링 매칭	열거	○	×	컴포넌트(UML)
Meta-Edit+[13]	클래스	스트링 매칭	열거	○	×	컴포넌트(UML)
Omnibuilder	패턴	스트링 매칭	객체	○	×	템플릿(UML)
ModelMaker	패턴	스트링 매칭	Gamma	○	×	템플릿(UML)
IBM설계패턴 라이브러리	패턴	스트링 매칭	Gamma	×	×	×
본 논문의 제시방법	패턴	E-SARM	Gamma+패싯	○	○	패턴(UML)

따라서 본 연구는 UML 모델링이 가능하고, 설계패턴을 컴포넌트화하여 패턴과 항목의 관계를 정의하여 패싯 분류하였으며, 패턴에 적용될 수 있는 최적의 패턴 참조 회수를 추출하여 유사한 패턴들까지 검색할 수 있는 설계패턴 검색 시스템을 구축하였다. 검색된 패턴들은 질의어에 대한 유사도 우선 순위로 나타나기 때문에 우선 순위가 큰 패턴 순서로 검색하여 선택하면 보다 빠르고 효과적인 방법일 뿐만 아니라 유사패턴까지 검색되기 때문에 사용자가 선택의 범위가 넓다는 장점을 갖고 있다. 또한 패턴 검색 비용이 직접 연결 검색보다 적게 들어가기 때문에 컴포넌트 확장시 보다 유리한 정보저장소를 구축할 수 있다.

6. 결 론

본 연구는 설계패턴이 가지고 있는 속성과 특징, 메타모델 등의 패턴 정보를 이용하여 설계패턴 검색 시스템을 구축하였다. 패턴 정보는 패턴의 분류와 검색을 위한 데이터를 가지고 있는 패턴 속성정보와 검색된 패턴을 뷰어에 UML로 표현하는데 필요한 패턴 메타정보로 구성된다. 본 시스템은 설계패턴의 효율적인 관리와 재사용을 위하여 패싯 방식을 이용한 패턴 분류와 패싯 항목과 패턴간의 관련성을 이용한 검색 방법을 사용하여 검색된 패턴을 UML 클래스 다이어그램으로 나타낼 수 있는 설계패턴 검색 시스템이다.

패턴 라이브러리는 패턴 속성정보가 저장되어 있는 패턴 데이터베이스와 패턴 메타정보가 저장되어 있는 메타 데이터베이스로 구분된다. 패턴 데이터베이스에는 Gamma의 패턴 표현 정보와 패턴의 패싯 분류 정보, 재사용 정보 등이 있으며, 이 속성을 바탕으로 기본 패턴과 추가 패턴을 분류하여 관리할 수 있도록 하였다. 또한 패싯 항목에 따라 분류된 패턴은 검색 시 패턴 속성정보를 이용하여 패턴과 항목 사이의 관련성을 구하여 관련 패턴을 검색할 수 있도록 해준다. 메타 데이터베이스에는 설계패턴 구조로부터 추출한 클래스 메타모델, 속성 메타모델, 오퍼레이션 메타모델, 관계 메타모델이 저장되어 있어, 설계패턴을 UML 클래스 다이어그램으로 표현할 수 있도록 하였다. 패턴 뷰어는 편집기와, 검색 인터페이스, 패턴 추가·삭제의 패턴 관리기 등의 인터페이스를 제공하여 효율적인 UML 모델링 도구를 지원해 준다. 또한 패턴 관리시스템은 신규 패턴의 등록과 유사도를 이용한 관련 패턴의 검색, 패턴 정보의 등록과 같은 역할을 수행하여 라이브러리와 패턴 뷰어 사이의 데이터 흐름과 처리를 담당한다.

설계패턴의 효율적인 관리와 검색을 위하여 Gamma의 분류 방법을 확장하여 패싯 분류하였고, E-SARM을 이용한 검색을 실시하였다. 패싯은 패턴을 구현하고 실행할 수 있는 실제 상황을 기술할 수 있도록 특정 응용 도메인 종속 여부와 패턴이 속한 영역, 적용 목적 그리고 적용 범위

등을 포함하도록 정의하였다. 질의어에 완전히 부합하는 패턴이 검색되지 않았을 경우에도 유사한 패턴의 검색을 지원할 수 있도록 패턴들 사이에 유사도를 측정하여 유사도가 높은 순서로 관련 패턴이 제공될 수 있도록 구현하였다. 이 방법은 검색 시간을 줄이는 동시에 질의 작성시 선택한 항목과 직접 연결된 패턴뿐 아니라 간접적으로 밀접하게 연결된 패턴들도 검색해 준다. 시뮬레이션 결과 패턴의 유사도까지 측정한 관련 패턴을 함께 제공함으로써 직접 연결 검색보다 선택의 폭이 평균 14% 더 넓었으며, 관련 패턴 검색이 직접 연결 검색 방법보다 패턴 선택 비용이 적다는 것을 알 수 있었다.

본 연구는 설계패턴의 패턴 정보를 이용하여 패턴이 실제 사용될 수 있는 경험적인 상황을 기준으로 패턴을 패싯 분류하고, 관련 패턴의 검색이 가능하여 재사용 패턴의 선택 폭을 넓힐 수 있는 설계패턴 검색 시스템을 구축하였다. 본 시스템은 기존의 시스템과 비교하여 Gamma 패턴 외에도 신규 패턴의 등록과 관리가 가능하며, 증가하는 패턴에 대한 효율적인 분류와 검색을 지원하고, 패턴이 UML로 표현되고 모델링될 수 있는 설계패턴 검색 시스템이다. 따라서 컴포넌트화된 패턴의 분류가 간단하고 관련 패턴의 식별이 용이하며, 패싯 항목을 선택하여 검색함으로써 질의 작성이 쉬울 뿐 아니라 UML 다이어그램을 사용함으로써 객체지향 방법론을 효과적으로 수행할 수 있는 검색 시스템이다.

그러나 본 연구는 웹을 기반으로 한 설계패턴의 검색과 다중 사용자에 의한 패턴 관리가 요구된다. 또한 검색 효율성의 극대화를 위해서 유사한 기능을 하는 패턴을 서로 가까운 위치에 저장시켜 라이브러리를 구축하는 것이 요구된다. 앞으로의 연구방향은 유사성을 패턴 구조적인 면에서 비교하여 중복저장의 문제를 해결하고 패턴 클러스터링을 통하여 효율적인 라이브러리를 구축하는데 있다.

참 고 문 헌

[1] http : //www.omg.org.

[2] http : //st-www.cs.uiuc.edu/~plop/.

[3] http : //www.argo.be/europlop/.

[4] Wojtek Kozaczynski, Grady Booch, "Component- Based Software Engineering," IEEE Software, Vol.15, No.5, pp. 34-36, Sept/Oct, 1998.

[5] http : //www.omnibuilder.com/.

[6] http : //www.modelmaker.demon.nl/.

[7] F.J. Budinsky, M.A. Finnie, J.M. Vlissides, P.S. Yu, "Automated code generation from design patterns," Object technology, IBM Systems Journal Vol.35, No.2, 1996.

[8] J. Suzuki, Y. Yamamoto, "Managing the Software Design Documents with XML," http : //www.yy. ics.keio.ac.kr/~suzuki, 1999.

[9] 한정수, 송영재, "개선된 SARM을 이용한 객체지향 부품 재사용 시스템", 정보처리논문지, 제7권 제4호, pp.1092-1102, 4. 2000.

[10] E.Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Pattern : Elements of Reusable Object -Oriented Software," Addison-Wesley, 1995.

[11] M. Pitteway, "On some pel level research problems", Proceedings of the 13th Symposium on Computer Arithmetic, 1997.

[12] http : //www.plasticsoftware.com/.

[13] http : //www.metacase.com/.

김 귀 정

e-mail : scallet@case.kyunghee.ac.kr

1994년 한남대학교 전자계산공학과(공학사)

1996년 경희대학교 전자계산공학과 (공학석사)

1999년 경희대학교 전자계산공학과 박사과정 수료

1997년~현재 경희대학교 전자계산공학과 박사과정

관심분야 : 소프트웨어공학, S/W 재사용, CASE도구

송 영 재

e-mail : yjsong@kyunghee.ac.kr

1969년 인하대학교 전기공학과(공학사)

1976년 일본 Keio University 전산학과 (공학석사)

1979년 명지대학교 대학원졸업(공학박사)

1971년~1973년 일본 Toyo Seiko 연구원

1982년~1983년 미국 Univ. of Maryland 전산학과 연구교수

1985년~1989년 IEEE Computer Society 한국지회부 회장

1984년~1989년 경희대학교 전자계산소장

1976년~현재 경희대학교 전자계산공학과 교수

1993년~1995년 경희대학교 교무처장

1996년~1998년 경희대학교 공과대학장

1998년~2000년 경희대학교 기획조정실장

2001년~현재 경희대학교 산업정보 대학원장

관심분야 : 소프트웨어공학, OOP/S, CASE 도구, S/W 개발도구론, S/W 재사용