디자인 패턴 구조를 이용한 클러스터링에 관한 연구

A Study on Clustering Algorithm Using Design Pattern Structure

한정수
천안대학교 정보통신학부 교수

김귀정
건양대학교 IT 학부 교수

Jung-Soo Han
Professor, Division of Information & Communication, Cheonan University
Gui-Jung Kim
Professor, Division of Information Technology, KonYang University.

중심어 : 설계패턴, 클러스터링, 클래스 다이어그램, 패턴검색

요 약

클러스터링은 부품 분류의 대표적인 방법인데, 클래스나 모듈의 응집도와 결합도를 이용한 기존의 클러스터링 방법은 클래스간의 관계에 중점을 둔 디자인 패턴을 기존의 클러스터링 방법을 이용하는 것은 효과적일 수 없다. 본 논문에서는 디자인 패턴을 분류하기 위해 패턴 구조의 특성을 가지고 분류하였다. 그리고 클러스터링에 의한 분류는 패싯 분류에 의한 방법보다 높은 정확도를 보여주었다. 따라서 자동화된 분류방법인 클러스터링 알고리즘을 사용하여 디자인 패턴을 분류하는 것이 효과적이라 할 수 있다. 디자인 패턴의 분류는 검색 시 유사한 패턴들이 같은 카테고리에 저장이 되므로 유사 패턴을 비교하여 사용할 수 있으며, 패턴 클러스터링에 의해 분류되고, 패턴의 링크정보를 이용하여 저장하므로 저장소를 효율적으로 관리 할 수 있다.

Abstract

Clustering is representative method of components classification. But, previous clustering method that use cohesion and coupling can not be effective, because design pattern has consisted by relation between classes. In this paper, we classified design patterns with special quality of pattern structure. Classification by clustering had expressed higher correctness degree than classification by facet. Therefore, can do that it is effective that classify design patterns using clustering algorithms that is automatic classification method.

When we are searching design patterns, classification of design patterns can compare and analyze similar patterns because similar patterns is saved to same category. Also we can manage repository efficiently because of using and storing link information of patterns.

I. 서론

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 디자인 패턴이다[1]. 디자인 패턴은 객체지향 방법론의 가장 큰 장점인 재사용성과 모듈성을 극대화시킨 실제 구현 과정에서의 해결책이며 현재 미국내의 설계 패턴 컨퍼런스인 PLoP(Pattern Languages of Programs)와 유럽의 설계 패턴 컨퍼런스인 EuroPLoP를 통해서 공식적으로 발표되어 알려져 있는 디자인 패턴은 수 백 가지에 이른다. 이렇게 증가하고 있는 패턴의 재사용성을 향상시키기 위해서 부품의 효율적인 관리가 필수적이다.

기존의 클러스터링은 클래스간의 ,클러스터링이나 클래스내의 클러스터링이 주를 이루었다. 따라서 기존의 클러스터링 방법은 클래스나 모듈의 응집도와 결합도를 이용하여 클러스터링 하였다[2]. 그러나 클래스간의 관계와 구조에 중점을 준 디자인 패턴을 이러한 응집도나 결합도를 이용한 기존의 클러스터링 방법으로 클러스터링 하는 것이 효과적일 수 없다. 따라서 본 연구는 클래스간의 관계로 이루어진 디자인 패턴을 클러스터링하기 위하여 패턴 클러스터링 알고리즘을 제안하였다. 패턴 클러스터링 과정은 패턴들이 가지고 있는 특성을 위해 2단계의 분류과정을 거치는데 1단계 클러스터링 과정에서

는 패턴의 기능에 따른 분류를 하며 2단계 클러스터링 과정에서는 패턴의 구조를 가지고 분류를 하게 된다. 패턴 구조에 의한 분류 시 패턴 알고리즘을 사용하여 분류한다. 제안한 디자인 패턴의 클러스터링은 패턴 저장 시 패턴 클러스터링에 의해 분류되어 패턴의 링크정보를 이용하여 저장하므로 저장소(Repository)를 효율적으로 관리할 수 있으며 시스템 구성도로부터 패턴의 사용정보를 추출, 이용하여 시스템 재설계시에 도움이 될 수 있도록 하였다.

II. 관련 연구

1. Gamma의 패턴 분류

디자인 패턴을 Gamma는 우선적으로 패턴이 하는 역할에 따라 생성패턴, 구조패턴, 행위패턴으로 구분하였다. 생성패턴은 객체의 생성방식을 결정하는 포괄적인 솔루션을 제공하는 패턴으로 클래스 정의와 객체 생성 방식을 구조화, 캡슐화 하는 방법을 제시한다. 구조패턴은 클래스와 객체가 보다 대규모 구조로 구성되는 방법에 대한 해결책을 제시하는 패턴으로 다른 기능을 가진 객체가 협력을 통해 어떤 역할을 수행할 때 객체를 조직화시키는 일반적인 방식을 제시한다. 마지막으로 행위 패턴은 객체의 행위를 조직화, 관리, 연합하는데 사용되는 패턴으로 객체간의 기능을 배분하는 일과 같은 알고리즘 수행에 주로 이용된다. 또한 표 1에서 보는 바와 같이Gamma는 패턴의 사용목적에 따라 24개의 기본적인 패턴을 정의하였다[3].

표 1. Gamma의 디자인 패턴 분류

Purpose		
Creational	Structural	Behavioral
Abstract Facotry Builder Factory Method Prototype Singleton	Adapter Brdge Composite Decorator Facade Flyweidght Proxy	Chain of Responsibility Command Interpreter Iterator Mediator Menento Obsever State Strategy Template Method Visitor

2. GTE사의 패싯 분류 시스템

Diaz에 의해 제안된 GTE사의 시스템은 패싯 분류 방법을 이용한 대표적인 재사용 시스템이다[4]. 패싯 분류 방법은 열거형 분류 방법의 단점인 부품의 확장성을 개선시킨 방법으로 부품들이 갖는 공통적인 측면의 특성을 합성하여 하나의 패싯으로 표현한 후 하나의 부품을 여러 개의 패싯으로 나타낸다. 이 분류 방법은 부품들이 갖는 기본적인 클래스만 표현하므로 분류가 간단하며 이해하기가 쉽고, 확장이 용이하다. 질의는 수정도 가능하며 검색에 실패할 경우 기능이 비슷한 부품의 추출을 위해 동의어 관리 방법을 이용하여 새롭게 질의가 작성된다. 그러나 한번 분류가 설계되면 고정적이 된다는 제한성을 가지며 패싯 항목이 많아지면 그들 사이에 관련성의 명세와 동의어 처리가 어렵고, 검색 시간이 길다는 단점이 있다.

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. 표 2는 도메인 종속 여부와 적용 범위의 패싯 특성과 Gamma의 분류를 패싯의 기본으로 나타낸 것이며 각 패싯의 특징을 표현하는 항목을 표 3과 같이 정의하였다.

표 2. 패싯의 기본 요소

패 싯		특 성
도메인 종속 여부		패턴 재사용 시 특정 도메인 종속 여부
Gamma 분류	영역	패턴이 속한 영역, 클래스나 객체와의 관련성
	목적	패턴의 역할에 따른 분류
적용범위		패턴의 실제 적용 범위

표 3. 각 패싯의 항목

패 싯		특 성
도메인 종속 여부		도메인 독립 도메인 종속
Gamma 분류	영역	클래스(class) 객체(object)
	목적	생성(creation)패턴 구조(structural)패턴 행위(behavioral)패턴
적용범위		요구 사항을 객체로 분할 시스템 기능을 객체로 분할 객체의 인터페이스 지정 객체 지향 프로그래밍 적용 델리게이션 메카니즘 적용 런타임, 컴파일타임 구조의 관계규정

3. 패턴 추출 위한 개념 분석

개념분석은 공통적인 속성을 가지는 객체를 그룹화 하는 것을 가능하게 한다. 개념 분석의 적용에 있어서 객체는 클래스의 그룹이고 속성(attribute)은 클래스 사이의 관계이다. 개념

분석의 첫 번째 관점은 상황(context), 즉 객체들의 집합, 속성들의 집합, 그리고 각각의 객체들이 가지고 있는 속성을 정하는 객체와 속성사이의 이진 관계이다. 지금의 적용을 위한 클래스 쌍의 이진관계는 각각의 관계에 의해 연결된다. 개념(concept)은 공통적인 속성을 가지는 객체를 최대한 모아 놓은 것이다. 즉 속성을 공유하는 모든 객체를 그룹화 한 것이다. 더 일반적으로 개념은 (X,Y)으로 되어 있다.

즉,

$$X = o \in O | \forall a \in Y ; (o, a) \in P \quad (1)$$

$$Y = a \in A | \forall o \in X ; (o, a) \in P \quad (2)$$

여기에서 O는 객체이고 A는 속성이다. P는 그들 사이의 이진 소유관계이다. X는 개념의 범위(extent)를 말하며 Y는 개념의 의미(intent)를 말한다. 개념 분석의 사용을 위한 주된 관점은 디자인 패턴이 일반적으로 개념에 해당된다. 사실상 개념은 공통적으로 관계를 공유하는 클래스 그룹으로 구성된다. 이와 같은 클래스 그룹은 속성의 수가 패턴의 복잡도를 결정하는 동안 코드나 설계에서 발견할 수 있는 패턴에 의해 주어진다. 패턴 후보로 고려되는 클래스 그룹의 크기는 디자인 패턴의 순서에 따른 정렬에 의한다. 그림 1의 클래스 다이어그램에서 클래스 사이의 관계는 클래스의 C×C 한 쌍과 클래스 관계R을 통하여 표현할 수 있다. 여기에서 C는 클래스 다이어그램 안에서 모든 클래스를 말한다. 즉 $(Y, X)_e \in R$은 Y와 X사이의 확장(extend)관계이고 $(H, K)_a \in R$은 H와 K사이의 연관(association)관계를 의미한다. 따라서 클래스는 순차배열(sequence)로 모여지게 된다. 각각의 포함된 클래스들은 이름에 의해서 보다 이것의 순차적인 인덱스에 의해 정렬된다. 그림 1의 참조에서 두개의 클래스 순차배열 (B,A,C)와 (Y,X,Z)는 공통적으로 $(1,2)_e$ 와$(1,3)_a$의 속성을 가진다. 사실상 순차배열의 첫 번째 클래스는 두 번째 클래스와 확장관계(extend)에 있고 순차배열의 첫 번째 클래스는 세 번째 클래스와 연관관계(associate)에 있다[5].

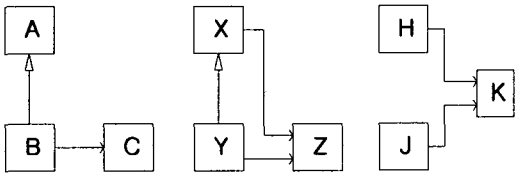

그림 1. 상속, 연관관계를 포함하는 다이어그램

III. 패턴 클러스터링

1. 패턴 클러스터링 과정

패턴 클러스터링 과정은 그림 2에서와 같이 크게 2단계로 나누어진다. 첫 번째 단계에서는 패턴이 어떠한 역할을 수행하는가의 역할에 따른 기능적 분류를 하고 두 번째 단계에서는 패턴들이 가지고 있는 클래스들 간의 관계를 이용하여 클러스터링 하는 구조적 분류 단계로 이루어져있다. 사용자는 UML의 클래스 다이어그램을 사용하여 사용자 패턴을 형성하면 클러스터링 과정을 통하여 패턴 라이브러리에 저장하게 된다. 그러나 패턴 구조를 이용한 클러스터링 방법은 단순히 패턴의 구조가 일치하는 부분이 있다고 해서 관련이 있는 패턴이라고 정의 내리기 어렵다. 따라서 패턴구조를 이용한 클러스터링 하기 전에 전 단계로 패턴을 기능적으로 분류하는 과정을 거친다. 기능적 분류 단계에서는 패턴이 하는 역할에 따라 패턴을 3가지로 분류한다. 객체의 생성 방식을 결정하는 포괄적인 방법을 제공하는 패턴이면 생성 패턴으로 분류하고, 객체를 조직화 시키는 일반적인 방식을 제시하는 패턴이면 구조패턴으로 분류하며, 그리고 객체간의 기능을 분할하는 역할을 수행하는 패턴이면 행위패턴으로 분류하여 인위적으로 선택을 한다. 기능적으로 분류된 것을 기준으로 구조적 분류단계에서는 패턴의 구조적인 형태를 이용하여 클러스터링하도록 하였다. 구조적 분류 단계에서의 클러스터링 방법으로는 3가지로 분류된 Gamma의 기본 패턴중의 하나와 사용자가 추가한 패턴이 구조가 일치하는 패턴을 같은 분류로 클러스터링 하도록 클러스터링 알고리즘을 사용하였다.

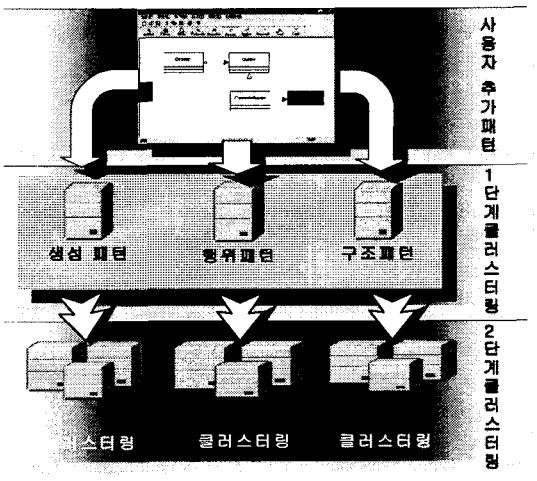

그림 2. 패턴 클러스터링 과정

2. 패턴 클러스터링 알고리즘

패턴 클러스터링 알고리즘은 기준으로 정한 패턴 중에서 사용자 추가 패턴의 구조와 일치하는 구조가 있는 경우에 같은 그룹으로 클러스터링 되도록 하는 알고리즘이다. 패턴의 구조는 UML의 클래스 다이어그램에서 클래스와 클래스의 관계로 나타내어진다. 패턴의 구조를 비교하기 위해 클래스와 클래스간의 관계를 하나의 순서쌍으로 하여 하나의 패턴을 순서쌍의 그룹으로 변환한다.

$$P = R_k(i,j) \mid (i,j) \in R,\ i \in C,\ j \in C,\ 1 \le k \le n \quad (1)$$

여기서 P는 패턴의 순서쌍을 나타낸다. R은 클래스의 관계를 나타내고 C는 클래스를 나타낸다. 그림 3은 프록시 패턴을 보여주며 이를 순서쌍으로 변환하면 P= {G(2,1),S(3,2)} 으로 표현된다. 각 하나의 순서쌍에서 앞부분의 영어는 클래스 다이어그램의 관계를, 숫자는 클래스를 나타낸다. 표 4는 클래스다이어그램에서의 관계를 나타내는 기호와 순서쌍에서의 약자 표시를 표로 나타내었다. 순서쌍의 뒤 부분은 관계를 가지는 두 클래스를 나타내었다. 또한 사용자 추가 패턴이 그림 4와 같다면 사용자 패턴을 순서쌍으로 변환하면 P=S(1,2), G(1,3), G(4,3), S(5,4), G(6,5), G(7,5)으로 변환할 수 있다. 추가 패턴은 프록시 패턴이 확장된 상태를 나타내고 있다. 이 패턴 안에서 프록시 패턴을 찾는 알고리즘은 두 패턴을 나타내는 순서쌍의 비교로써 이루어진다.

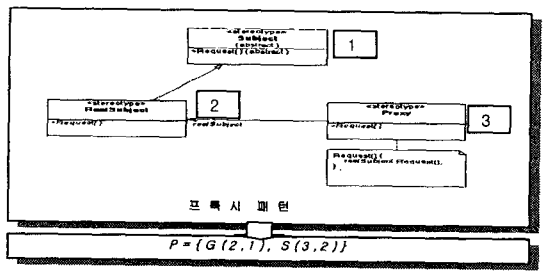

그림 3. 프록시 패턴의 순서쌍 표현

표 4. 클래스의 관계 표시

관 계	약 자
연관관계(Association)	S
일반화관계(Generalization)	G
집합연관관계(Aggregation)	E
복합연관관계(Composition)	D
의존관계(Dependency)	C
실체화관계(Realization)	R

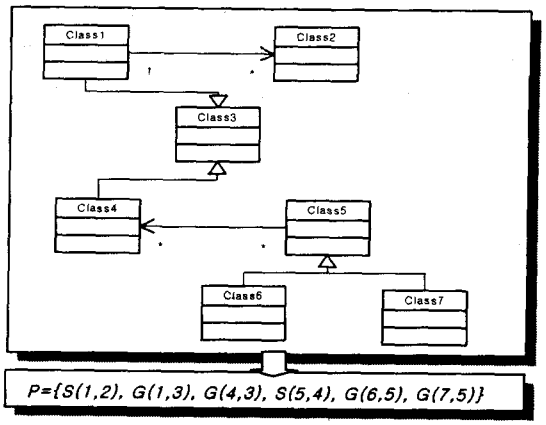

그림 4. 사용자 추가 패턴 순서쌍 표현

순서쌍으로 표현된 기본패턴과 사용자 추가 패턴을 비교하기 위하여 먼저 숫자로 표현된 클래스를 변환해 주는 과정을 거친다. 그림 5는 패턴 순서쌍을 변환하는 알고리즘을 나타낸다. 변환 이유는 클래스의 숫자는 임의적인 숫자를 사용하기 때문에 순서쌍이 놓이게 되는 순서가 바뀔 때마다 서로 비교가 되지 않고 다른 결과를 나타내는 것을 방지하고 순서쌍이 놓이는 순서와 상관없이 비교하기 위한 과정이다. 변환과정은 처음에 놓이는 순서쌍을 임의의 문자로 바꾼다. 여기서는 클래스를 나타내는 첫 번째 순서쌍의 숫자로 나타낸 첫 번째 클래스는 a로 바꾸고 두 번째 클래스는 b로 바꾸어준다. 나머지 순서쌍에서 첫 번째 순서쌍의 첫 번째 클래스를 나타내는 숫자와 같은 숫자는 a로 바꾸어 주고 첫 번째 순서쌍의 두 번째 클래스를 나타내는 숫자와 같은 숫자는 b로 바꾸어 준다. 순서쌍에서 변경된 클래스와 쌍을 이루는 클래스는 어떠한 숫자로 표현된 클래스라도 비교에 있어 큰 의미가 없으므로 x로 표현하여 준다.

Initial Step $P_1 = \{ R_1=(i_1, j_1) \mid a \to i_1,\ b \to j_1 \}$

Step $2 \le k \le n$

$P_1 = \{ R_k(i_k, j_k) \mid i_k \neq i_1$ 이면, $i_k \to a,\ j_i \to x$

$j_k \neq i_1$ 이면, $i_k \to x,\ j_i \to a$

$j_k \neq j_1$ 이면, $i_k \to x,\ j_i \to b$

$i_k \neq j_1$ 이면, $i_k \to b,\ j_i \to x \}$

그림 5. 패턴 순서쌍 변환 알고리즘

P= G(1,3), G(4,3), G(6,5), G(7,5), S(1,2), S(5,4)

⇒P=G(a,b), G(x,b), G(6,5), G(7,5), S(a,x), S(5,4)

위에서처럼 첫 번째 순서쌍의 클래스를 (a,b)로 변환하고 1은 a로 변환되고 3은 b로 변환되었기 때문에 나머지 클래스 중에서 1은 a로 3은 b로 변환하고 a, b로 변환된 순서쌍과 쌍을 이루는 클래스는 비교에 있어 어떤 숫자를 나타내는 클래스가 와도 상관없으므로 x로 변환하였다.

```
//기본패턴 변환
Pf={Pf1,Pf2,Pf3,....Pfk) | Transformation algorithms
                          of Foundation Pattern }
// 패턴 순서쌍 비교를 위한 루프
If(i=0 ; i<n ; i++)
 //비교를 위해 추가 패턴을 변환
  Pf={Pa1,Pa2,Pa3,....Pak) | Transformation algorithms
                          of Foundation Pattern }
//패턴을 비교해서 포함관계이면 저장하고 루프를 나온다
  Pf ⊂ Pa : Break Addition();
// 패턴구조가 불일치면 순서쌍의 순서를 바꾸어서 비교
  Pa1 → Pan, Pak→Pa(k-1)
Endif
```

그림 6. 패턴 비교 알고리즘

그림 6에서 비교 패턴 알고리즘에서 P_f는 변환후의 기본패턴을 나타내며 P_a는 변환후의 추가패턴을 나타낸다. 사용자가 추가한 패턴이 기본패턴과 비교하기 위해서 우선 기본패턴을 변환 알고리즘을 사용하여 변환한 다음 추가 패턴 또한 변환 과정을 거쳐 두 패턴이 같은 구조를 가지고 있는지 비교한다. 이 때 추가 패턴의 관계를 나타내는 여러 개의 순서쌍 중에 기본패턴과 관련이 있는 패턴을 찾아내기 위해 추가 패턴의 관계를 나타내는 순서쌍을 차례대로 변환과정을 거친다. 사용자 추가 패턴의 관계를 나타내는 순서쌍을 하나씩 변환과정을 거쳐서 아래 조건 식(2)에 만족하면 추가패턴 P_a는 기본 패턴 P_f에 클러스터링된다. 모든 순서쌍을 변환하였는데도 조건 식(2)에 만족하지 않으면 다른 기본패턴과 비교하는 과정을 거친다.

$$P_f \subset P_a \qquad (2)$$

그림 7은 사용자 추가 패턴과 프록시 패턴과의 비교 과정을 구체적으로 도식화 한 것이다. 프록시 패턴은 하나의 연관관계와 상속관계로 이루어져있다. 사용자 패턴 중에 같은 구조를 가지는 연관관계와 상속관계를 찾아내기 위해서 프록시 패턴을 나타내는 G(1,3)을 G(a,b)로 변환하여 준다. 그리고 다른 순서쌍에도 1은 a로 3은 b로 변환을 모든 해준다. 변환후의 프록시 패턴의 순서쌍은 G(a,b), S(x,b)로 변환된다. 사용자 패턴도 변환 과정을 거쳐 비교를 하는데 사용자 패턴은 4개의 상속관계를 가진다. 이중에 실제적으로 프록시 패턴의 구조를 나타내는 상속관계를 찾기 위하여 4개의 상속관계 중 하나를 변화과정을 거쳐 프록시 패턴의 변환된 순서쌍과 비교한다. 이 변환과정을 거쳐 G(a,b),S(x,b)구조를 가지는 순서쌍을 포함하고 있으면 사용자 패턴은 프록시 패턴의 구조를 가지는 그룹으로 클러스터링된다. 4개의 상속관계를 나타내는 순서쌍을 비교하여 G(a,b),S(x,b)구조를 가지는 패턴이 나타나지 않으면 프록시 패턴과 사용자 추가 패턴은 같은 구조를 갖는 패턴이라 할 수 없고 다른 패턴과의 비교과정을 반복한다.

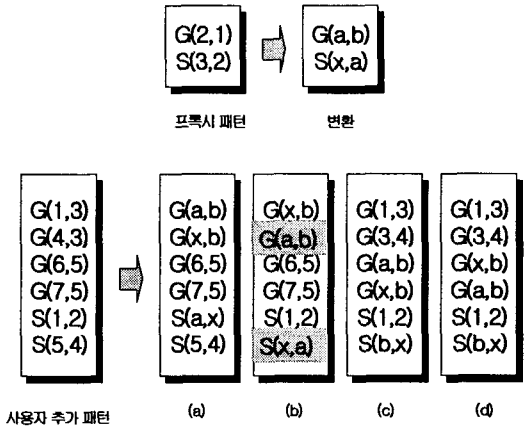

그림 7. 패턴 비교 과정

IV. 패턴 등록 및 관리 시스템

1. 패턴정보 데이터베이스

디자인 패턴의 정보는 패턴 검색 시에 패턴에 대한 정보를 확인할 수 있도록 설계하였다. strCargegory는 패턴의 3가지의 기능적 분류에 대한 생성패턴, 행위패턴과 구조패턴 중에 나타나도록 하였으며 docDocument는 패턴을 적용시킬 때 인식해야 하는 함정, 힌트, 기술 등 패턴에 대한 전반적인 설명을 위한 필드이다. strPart는 클러스터링에 의해 분류된 패턴이 속한 기본패턴을 나타내며 UserID는 패턴을 생성하여 등록한 사용자에 대한 아이디를 나타낸다. 표 5는 패턴정보 데이터베이스에 대한 테이블 구조를 나타낸다.

표 5. 패턴정보 데이터베이스의 테이블 구조

필드 이름	데이터 형식	설명
strName	varchar	패턴에 대한 이름
strCartegory	varchar	기능적 분류에 의한 카테고리
docDocument	text	패턴에 대한 설명
strPart	varchar	패턴이 분류된 기준
UserID	varchar	패턴 등록자 이름

2. 사용자 인터페이스

사용자는 패턴을 작성하여 패턴 라이브러리에 등록 할 수 있으며 패턴 DB로부터 패턴들을 검색할 수 있다. 그림 8에서와 같이 메인 화면에서 패턴을 UML 다이어그램을 사용하여 작성한다. 패턴이 작성되면 패턴을 등록하기 위하여 메뉴 중에 패턴항목을 선택한다. 패턴항목에는 패턴등록과 패턴DB 항목으로 구성되어 있다. 패턴 등록항목을 선택하면 패턴을 비교하기 위한 순서쌍을 표현하는 대화상자가 나타나는데 여기서 패턴의 구조를 보고 관계와 클래스가 표현된다. 또한 패턴이 나타내는 기능적 분류 중에 하나를 선택한다. 그러면 패턴명에 패턴설명 등 패턴에 대한 정보를 입력할 수 있는 대화상자가 나타난다. 그림 9의 패턴등록 대화상자에서는 패턴명에 패턴의 이름을 입력해주고 패턴 설명에는 패턴에 대한 간단한 문서를 작성하면 내부적으로 패턴 클러스터링 과정을 거쳐 패턴 라이브러리에 저장된다. 패턴이 클러스터링 과정을 거쳐 분류된 상태를 확인하거나 패턴들의 구조를 검색하기 위해서는 패턴DB항목 기능을 이용한다. 그림 10에서처럼 패턴DB는 패턴이 클러스터링에 의해 분류된 상태를 보여준다. 패턴은 Gamma의 24개의 패턴을 기준으로 분류되었는데 패턴의 구조를 보기 위해서는 해당되는 Gamma의 패턴 중에 클러스터링된 패턴들의 리스트가 나타난다. 이때 같은 카테고리에 있는 다양한 패턴들을 비교할 수 있으며 패턴을 선택하면 다양한 패턴에 대한 상세한 정보를 볼 수 있다. 패턴DB의 리스트 중에 하나를 선택하면 그림 11에서와 같이 패턴설명 대화상자가 나타난다. 패턴설명 대화상자에서는 패턴에 대한 상세한 정보를 확인 할 수 있다. 다음버튼이나 이전버튼을 선택하면 패턴DB보기에서의 콤보박스 내의 리스트에 있는 다른 패턴을 확인할 수 있기 때문에 사용자는 유사한 패턴들을 서로 비교 하여 정보를 확인 할 수 있다. 또한 패턴 구조 버튼을 선택하면 사용자는 패턴에 대한 구조적인 정보를 확인하여 재사용할 수 있도록 하였다.

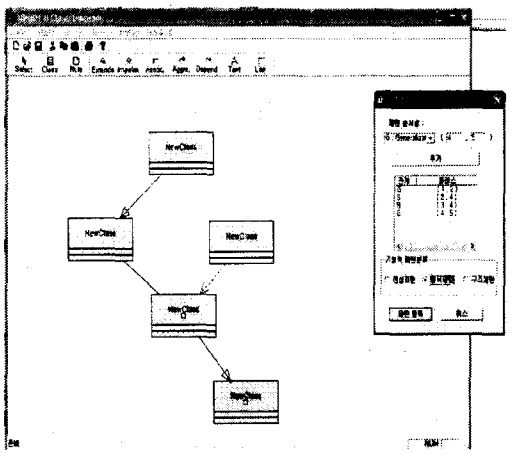

그림 8. 패턴 메인 화면

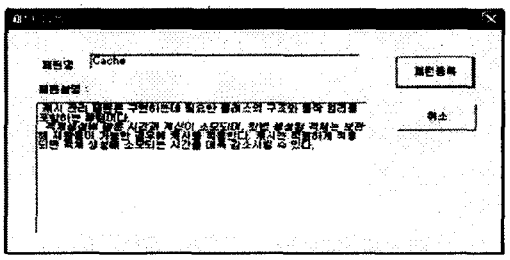

그림 9. 패턴 등록

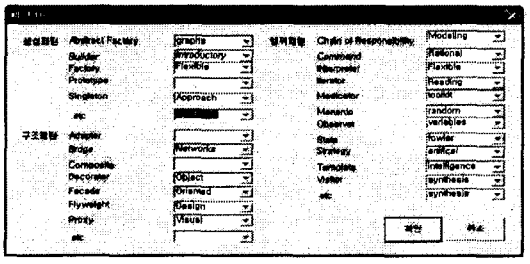

그림 10. 패턴 DB

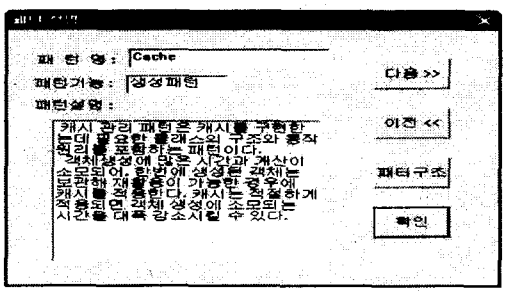

그림 11. 패턴 설명

V. 성능 평가

본 논문은 성능 평가를 위해 PLOP(Pattern Languages of Programs)컨퍼런스에 발표된 30개의 패턴을 사용하였다. 30개의 패턴을 본 논문에서 제시한 방법으로 분류하였고, 패싯 방법으로도 분류하였다. 패싯방법은 부품들이 갖는 공통적인 특성을 여러개의 패싯 항목으로 나타낸 후 부품의 특성이 가지고 있는 패싯들의 항목에 대하여 분류하는 방법[으로 패싯 분류에 대한 항목은 관련연구의 표 2, 표 3에 따라 분류하였다. 제안한 방법에 의해 정확하게 분류되었는지는 클러스터링에 의한 분류된 결과와 패턴의 실제정보와의 관련성을 파악하여 정확성의 유무를 판단하였다. 패싯 방법에 의한 분류는 패턴의 특성을 패싯항목 중에 선택하여 분류하도록 하였다. 표 6은 패턴들을 제안한 방법과 패싯분류 방법으로 분류한 후 관련성의 유무를 측정하였다. 표 7은 표 6을 기준으로 측정한 결과 제안한 방법의 분류는 90%정도의 정확도를 나타냈으며 패싯분류 방법에 의한 분류는 83.3%정도의 정확도가 측정되었다. 이는 추상화 개념인 패턴의 특성을 가지고 패싯 항목으로 패턴을 분류하는 것은 본 논문에서 제시한 구조를 이용하여 비교한 것 보다 낮은 결과가 나온다는 것을 알 수 있다. 또한 패싯 분류에 의한 방법은 패싯 항목의 선택이 패턴의 특성에 따른 임의적인 방법이기 때문에 정확한 패싯 항목의 선택여부에 따라 정확한 분류의 정도에 차이가 있을 수 있다.

또한 패턴 클러스터링에 의한 분류는 패턴들이 기본패턴에서 확장한 형태로 분류가 되기 때문에 확장된 패턴은 패턴구조를 저장하고자할 때 같은 분류 안에 있는 패턴들은 기본패턴에 대한 연결정보만 있으면 기본패턴 만큼의 중복되는 클래스만큼의 저장소의 크기를 줄 일 수 있다. 표 6은 정보저장소에 저장된 클래수의 수를 나타낸 것이고 그림 12는 그에 따른 정보저장소의 크기를 그래프로 비교한 것이다. 표 9는 검색 모델, 분류방법, 유사패턴 검색, 모델링 도구지원 등의 특성을 기준으로, 본 연구에서 설계한 설계 패턴 검색 시스템과 기존의 시스템과 비교하였다. CASE 도구로 많이 사용되고 있는 ModelMaker[6], Omibulder[7], Plastic[9]과 MetaEdit+[9]를 비교하였다. Plastic이나 MetaEdit+는 UML 모델링은 가능하지만 독립된 검색 모델이나 분류모델이 존재하지 않기 때문에 컴포넌트의 관리와 검색이 효율적으로 이루어지지 못한다. 또한 패턴 기반 시스템인 Omibulder와 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 6. 사용된 패턴들의 분류

순번	패턴	클래스 수	제안한 방법의 카테고리	분류	패싯분류의 카테고리	분류
1	Order/Shipment Pattern	6	memento	o	memento	o
2	The Stock Manager analysis Pattern	8	prototype	o	prototype	o
3	Rule Object[PLOP2000]	13	mediator	o	mediator	o
4	The Object Filter and Access Control Framework	18	strategy	o	strategy	o
5	Legacy Wrapping[PLOP2000]	6	adapter	o	adapter	o
6	modifier[PLOP2000]	8	prototype	x	factory method	x
7	Protocal System Architecture	15	bridge	o	proxy	x
8	Strategized Concurrency	5	state	o	state	o
9	Reductor[PLOP2000]	9	command	o	command	o
10	Phrasebook Pattern	4	builder	o	builder	o
11	Two Phase Commit[PLOP99]	7	decorator	o	decorator	o
12	Composite Trasaction[PLOP99]	7	composite	o	composite	x
13	Matcher-Handler[PLOP99]	6	obsever	o	obsever	o
14	Authemtcator Pattern	5	prototype	x	Builder	x
15	Alternator[PLOP99]	6	state	o	state	o
16	Mayfly[PLOP99]	10	interator	o	interator	o
17	ACEE architecture[PLOP99]	5	obsever	o	obsever	o
18	Abstract Machine[PLOP99]	11	abstract factory	o	abstract factory	o
19	Grafcet Pattern[PLOP99]	5	command	x	interator	o
20	Composite Calls[PLOP99]	10	composite	o	composite	o
21	Data Filter Architecture	9	proxy	o	proxy	o
22	Emissary Pattern[PLOP98]	5	strategy	o	state	x
23	Courier[PLOP98]	6	mediator	o	mediator	o
24	Override Current Processing	5	command	o	command	o
25	Substitution Pattern[PLOP98]	5	strategy	o	strategy	o
26	Delegation Pattern[PLOP98]	4	proxy	o	proxy	o
27	Reliable Hybrid Pattern	16	composite	o	composite	o
28	The DynamicTemplatePattern	6	observer	o	observer	o
29	Distributed Proxy[PLOP97]	9	proxy	o	proxy	o
30	Virtual Proxy[Larman98]	6	proxy	o	proxy	o

따라서 본 논문에서 제한 방법은 UML 모델링이 가능하고, 클러스터링에 의한 분류로 자동화된 분류방법을 사용하였고 검색 시에도 스트링매칭에 의한 검색과 같은 카테고리 내에서의 유사패턴 검색을 병행할 수 있도록 하였다.

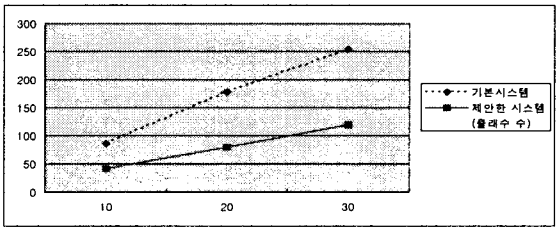

그림 12. 패턴의 정보저장소의 크기 비교

표 7. 분류된 패턴의 정확도 측정

패 턴	제안한 방법의 카테고리	패싯분류에서의 카테고리
정확도	90%	83.33%

표 8. 패턴 정보저장소의 크기에 따른 클래스의 수

저장소 \ 패턴의 수	10	20	30
기본시스템	92	164	235
제안한 시스템(클래수 수)	48	80	110

표 9. 기존의 시스템과 비교

	modelmaker [6]	omibuilder [7]	plastic 3.0 [8]	meta-edit+ [9]	본 시스템
컴포넌트	패턴	패턴	클래스	클래스	패턴
검색모델	스트링매칭	스트링매칭	스트링매칭	스트링매칭	스트링매칭
모델링 도구지원	템플릿 (UML)	템플릿 (UML)	컴퍼런트 (UML)	컴퍼런트 (UML)	패턴(UML)
패턴/컴포넌트 추가	■	■	■	■	■
확장검색 (유사패턴)	■	■	■	■	■
분류방법	Gamma	객체	열거	열거	클러스터링 분류

VI. 결 론

본 논문은 패턴의 효율적인 관리를 위해서 패턴의 구조를 이용한 패턴 클러스터링 알고리즘을 제안하였고, 패턴을 생성, 등록 저장, 검색할 수 있도록 구현하였다. 시스템에 적용 가능한 패턴을 등록하여 저장소에 저장하려고 할 때 우선 패턴 편집기를 이용하여 패턴의 구조를 만든다. 패턴 구조에 따라 패턴 구조와 정보를 저장하면 패턴 구조를 이용하여 패턴 클러스터링 알고리즘에 의해 23개의 카테고리 중에 하나로 분류되어 저장되도록 하였다. 이때 패턴 클러스터링 알고리즘은 패턴 분류를 위해 패턴이 가지고 있는 클래스의 구조를 이용하여 분류하는 방법으로서 추가패턴을 패턴의 클래스와 클래스의 관계를 하나의 순서쌍으로 나타내어 순서쌍으로 나타낸 기본패턴과 비교하여 일치하는 부분이 있을 경우에 한 그룹으로 클러스터링 되도록 한 것이다. 또한 카테고리 안에서 분류된 패턴을 검색하여 관련이 있는 유사패턴 검색이 가능하도록 하였다.

디자인 패턴은 기존의 클러스터링 방법으로는 효과적이지 못하므로 디자인 패턴 분류를 위한 패턴 클러스터링 알고리즘을 제안하였다. 패턴 클러스터링의 의해 분류한 결과 패턴의 구조에 의한 분류가 패턴의 추상적임 개념을 가지고 항목을 선택해서 분류하는 패싯 분류에 의한 분류에서보다 높은 정확도가 나타났으며 패턴 클러스터링에 의한 분류는 패턴 저장 시 다수의 중복되는 패턴을 패턴의 연결 정보만으로 중복되는 클래스 수만큼의 정보저장소 절약을 가져왔다. 또한 유사패턴을 비교할 수 있도록 하여 정보저장소를 효율적으로 관리할 수 있으며, 패턴에 대한 정확한 정보가 없는 패턴에 대해서도 분류할 수 있도록 하여 패턴에 대한 대략적인 정보를 유추할 수 있었다. 그리고 시스템 리엔지리어링 시 클래스로 표현된 전체 시스템 구성도로부터 디자인 패턴의 사용정보를 추출하기 때문에 시스템의 설계나 유지보수에 도움을 줄 수 있다.

향후 연구 방향으로 사용자 추가 패턴 중에 2개 이상의 기본패턴을 포함하고 있을 때의 확장된 클러스터링 방법과 패턴 분류 시 기존의 기본패턴인 Gamma의 기본패턴에 분류되지 않았을 경우 기본패턴을 확장하여 사용자가 기본패턴을 추가할 수 있도록 하는 다양한 분류 기준으로 분류할 수 있는 방법이 요구된다.

참 고 문 헌

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

[2] Nicolas Anquetil and timothy c. Lethbridge, "Experiments with Clustering as a software Remodularization Method", Proceedings of the 6th Working Conference on Reverse Engineering , 235-255 , 1999.

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

[4] R.Prieto-Diaz and P.Freeman, "Classifying Software for Reusability," IEEE Software, Vol.4, No.1, pp.6-16, Jan. 1987.

[5] Paolo Tonella and Giulio Antoniol, "Object Oriented Design Pattern Inference", Proceedings of the IEEE International Conference on Software Maintenance , 230-238 ,1999.

[6] www.modelmaker.demon.nl/

[7] www.omnibuilder.com/

[8] www.pasticsoftware.com/

[9] www.metacase.com/

한정수(Jung-Soo Han) 정회원

1990년 : 경희대학교 전자계산공학과 (공학사)

1992년 : 경희대학교 전자계산공학과 (공학석사)

2000년 : 경희대학교 전자계산공학과 (공학박사)

2001년 ~ ~현재 : 천안대학교 정보통신학부 교수

<관심분야> : 소프트웨어공학, S/W 재사용, CASE도구, CBSE

김귀정(Gui-Jung Kim) 정회원

1994년 : 한남대학교 전자계산공학과 (공학사)

1996년 : 한남대학교 전자계산공학과 (공학석사)

1999년 : 경희대학교 전자계산공학과 (박사수료)

2001년~현재 : 건양대학교 IT학부 교수

<관심분야> : 설계패턴, S/W 재사용, CASE도구