Analogy를 기반으로 한 분석/설계 패턴의 재사용

정 란† · 김 정 아†† · 김 행 곤†††

요 약

소프트웨어 재사용에서 코드 재사용은 다른 개발자에 의해 작성된 재사용 코드의 이해와 검색의 어려움 등 한계점을 갖는다. 코드 재사용 한계를 극복하기 위해서는 코딩 단계 이전의 설계 단계나 분석 단계의 산물을 재사용 할 수 있어야 한다. 본 연구에서는 객체 모델링 단계에서 이전의 경험을 재사용할 수 있는 환경을 제공하기 위해 질의와 컴포넌트에 대한 analogy를 판단하여 라이브러리의 모델과 패턴을 재사용할 수 있는 방법을 제안하였다. Analogy 매칭 기법은 설계 패턴의 검색, 이해 그리고 조합 등에 적용되며 본 논문에서 연구된 내용은 다음과 같다.

1. 재사용 라이브러리로부터 유사 컴포넌트 검색 및 추출을 위한 analogical 매칭 함수 제안
2. Analogical 매칭을 위해 라이브러리 내에 저장될 재사용 컴포넌트 표현 및 정의
3. 특성 도메인에 대한 의미적 정보의 라이브러리 설계
4. 사용자를 위한 analogical 매칭 에이전트 구축

Reuse of Analysis/Design Patterns Based on Analogy

Rhan Jung† · Jeong-Ah Kim†† · Haeng-Kon Kim†††

ABSTRACT

Code reuse in software reuse has several limitations such as difficulties of understanding and retrieval of the reuse code written by other developers. To overcome these problems, it should be possible to reuse the analysis/design information than source code itself. In this paper, we present analogical matching techniques for the reuse of object models and patterns. We have suggested the design patterns as reusable components and the representation techniques to store them. The contents of the paper is as follows.

1. Analogical matching functions to retrieve analogous components from reusable libraries.
2. The representation of reusable components to be stored in the library in order to support the analogical matching.
3. The design library with more semantic informations about domains.
4. The analogical matching agent with a user-friendly interface that can retrieve the analogous components from the library based on analogical matching techniques.

1. 서 론

소프트웨어의 재사용은 이전의 개발 경험을 새로운 소프트웨어 개발 과정에 재적용 하는 것으로 소프트웨어 개발 환경 및 관리 과정에서 생산성 향상에 기여할 수 있다[15]. 지금까지의 소프트웨어 재사용을 위한 노력으로는 대개 코드 수준의 재사용으로서 기존의 함수 라이브러리를 재사용 하거나 클래스 라이브러리를 재사용하는 것이었다[10]. 코드 재사용의 한계[2,3]를 극복하기 위한 방법으로 코딩 단계 이전의 분석/설계 단

※ 본 연구는 97년도 학술진흥재단의 공모과제 연구비에 의하여 연구되었음.
† 종신회원 : 삼척대학교 컴퓨터공학과 교수
†† 정 회 원 : 관동대학교 컴퓨터교육과 교수
††† 종신회원 : 대구효성가톨릭대학교 컴퓨터공학과 교수
논문접수 : 1998년 1월 15일, 심사완료 : 1998년 12월 28일

재에서 만들어진 산출물을 재사용 하려는 노력이 계속 되고 있다.

최근, 객체지향 방법론이 활성화됨에 따라 객체지향적 방법을 적용하여 소프트웨어를 개발하고자 하는 시도가 증가하고 있다[2,10]. 객체 모델링 경험이 없는 개발자들에게는 자신에게 주어진 문제에서 적절한 객체를 식별하고 정확한 속성과 행위를 부여하는 것이 쉽지 않다[11]. 그러나 유사한 분야에서 이미 작성된 모델을 참조할 수 있다면, 문제에 대한 개념을 쉽게 이해할 수 있고 이를 바탕으로 새로운 모델을 구축할 수 있을 것이다. 따라서, 환경과 언어에 종속적인 코드의 재사용보다는 개발 단계의 경험과 추상적인 모델을 재사용할 수 있도록 지원할 필요가 있다.

코드 재사용에는 코드의 기능적 특성을 표현하는 키워드를 중심으로 한 키워드 기반 매칭 기법이 적용될 수 있지만, 모델은 분석가에 따라 동일한 개념이라도 서로 다른 용어로 모델링 할 수 있고 서로 다른 개념을 응용 영역에 따라 같은 용어로 모델링 할 수도 있기 때문에 코드 재사용 기법과는 달라져야 한다.

본 논문에서는 요구 명세 단계에서 객체 모델의 재사용을 지원하기 위해 지식 공학에서 연구되어 온 analogy 기법을[4,7,12] 도입하여 analogical matching을 통한 객체 모델의 재사용 환경을 제안하고자 한다. Analogy 기법은 기존의 코드 재사용에 적용한 유사도(similarity) 측정 방식과는 차이가 있다. 유사도란 두 대상의 특성이나 본질들 사이에 공통되는 성질이 존재함을 의미하지만, analogy란 두 대상들간에 어떠한 관점에서든 서로 일치하는 구조나 대응하는 성질이 존재함을 의미한다[15]. 본 논문에서는 객체 모델의 재사용을 지원하기 위해 analogical matching 기법을 도입하여 analogy에 기반한 객체 모델과 패턴(pattern)의 재사용 환경을 제안한다. 이를 위해 객체 모델에 대한 라이브러리 표현 기법을 정의하고, analogical matching을 통한 검색 기법을 정의하고 이를 기반으로 한 검색 엔진을 개발하고 설계 라이브러리 구조를 정립한다.

2장에서는 재사용을 위해 필요한 기법을 기술하고, 3장에서는 모델의 재사용을 위한 라이브러리의 구조와 표현 기법을 정의하고 analogical matching 기법을 기반으로 한 패턴의 구문적 매칭 기법과 의미적 매칭 기법을 정의한다. 4장에서는 구축한 재사용 라이브러리로부터 주어진 상황에 적합한 모델을 검색하는 매칭 에이전트(agent) 구현에 대해 기술하고 이에 대한 평가를 했다. 5장에서는 결론과 향후 연구 방향에 관하여 기술하였다.

2. 모델 재사용을 위한 기존 연구

지금까지의 모델 재사용의 연구는 크게 명세(Specification)의 재사용과 기능중심의 산출물(예, DFD)의 재사용과 객체 모델의 재사용으로 크게 구분할 수 있다. 대부분의 연구는 산출물의 구조적 관련성을 바탕으로 Tree 형태의 내부 표현 기법을 바탕으로 기존의 SME(Structure Matching Engine), ACME(Analogical Constraint Matching Engine)을 사용하고 있다. 이를 위해 모델의 지식 표현 구조를 함수적 언어로 표현하여 SME와 ACME의 입력물로 활용하고 있다. 또한 SME와 ACME에서 만들어진 매칭 결과를 객체 모델의 재사용을 위해 해석하고 이 과정을 지원하는 도구들을 개발하였다. 본 논문에서는 가장 대표적인 Maiden의 접근법[9]과 Whitehurst의 접근법[15]에 관하여 기술하였다.

2.1 Maiden의 접근법

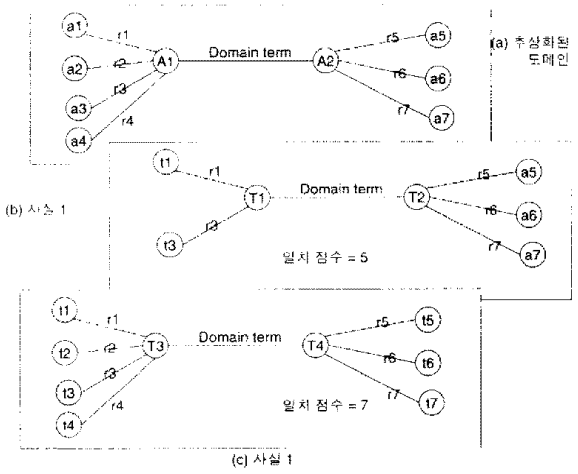
기존에 이루어진 Faceted 분류 기법이나 템플리트에 의한 재사용을 탈피하고자 구문적 정보에 기반하지 않고 의미적 지식에 기반한 재사용 지원 환경을 제안하였다. Maiden은 4가지 유형의 소프트웨어 개발 과정에서 일어나는 개발자들의 행위를 분석하여, 소프트웨어 개발 과정 중에 각 산출물들의 analogical 재사용에 필요한 내용을 도구로 개발하였다.

Maiden의 연구에서는 도메인에 대한 기본 지식을 최대한 활용하기 위해 도메인 지식에 대한 추상화 작업이 강조되었다. 도메인의 추상화 작업의 결과로 객체의 구조와 도메인에서의 객체의 유형과 시스템 행위를 구성하는 행동(action)과 행동 수행을 위한 상태 전이 개념으로 도메인에 대한 지식 베이스를 구축하였다. 정의된 개념은 구조적 응집성 알고리즘(Structural Coherence Algorithm)의 입력 형태의 조건식(Predicate)으로 변형되어진다. 매칭은 다음 6가지에 대해 이루어진다.

<표 1>의 표현 구조를 바탕으로한 매칭 함수를 정의하였다. 매칭 함수의 기본 개념을 그림으로 설명하면 다음과 같다.

〈표 1〉 Maiden이 제안한 Analogical Matcher
〈Table 1〉 Analogical Matcher by Maiden

객체 구조	<object, object, strucutral relation>
도메인요구 사항	<object, object, structural-relation, value>
상태전이	<object, source, destination, transition>
객체유형	<object, object-type>
상태전이 조건	<precondition, object, source, destination, transition>
외부시스템 자극	<state transition>
전이를 반추하는 함수	<function>



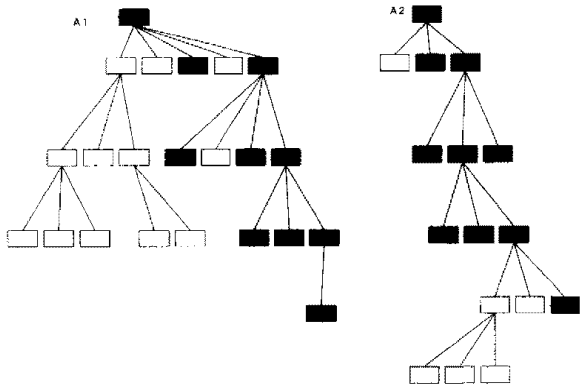
(그림 1) 구조적 응집성 알고리즘의 예
(Fig. 1) Example demonstrating the structural coherence algorithm

위의 예에서 추상화된 도메인(a)에 대해 개발자가 입력한 사실을 매칭한 결과 (b)의 경우는 T1과 T2에 연결된 관련성 정보 중 r1, r2, r5, r6, r7이 일치하므로 5점이고 (c)의 경우는 7이 된다. 그러므로 사실 2가 추상화된 도메인 모델에 더 유사한 것으로 판단한다. 즉 앞에서 정의한 지식 기반을 바탕으로 이를 도메인 지식을 중심으로 관련된 속성으로 정의한후 각 속성별 유사 정도를 판단하여 일치 정도를 판단한다.

2.2 Whitehurst의 접근법

지금까지 소프트웨어의 재사용이 코드 중심의 재사용이었기 때문에 관리적, 기능적 제한점을 해결하기 위해서는 보다 추상적이고 일반적인 산출물을 바탕으로 대형 소프트웨어 라이브러리를 구축해야 하며, 이를 지원하는 도구의 개발이 필요하다. Whitehurst의 이를 해결하기 위해 analogical reasoning 개념에 기반하여 소프트웨어 산출물에 대한 기본 개념과 그들간의 관련성에 대한 가정(assertion)을 정의하록 하였다. 이 가정을 통해 소프트웨어에 대한 의미적 정보를 기술하고자 하였다. Whitehurst의 접근법에서는 객체 자체의 성질보다는 객체들간의 관련성을 정형화하여 표현하는데 주안점을 두었다. 즉 패턴과 프레임워크의 재사용에 필요한 지식 표현 기법을 제안한 것이다. 상속이나 집합 관계의 관련성은 모두 연결 관계로 일반화하여 정의하도록 하고 있다.

매칭 알고리즘은 Strucutral Mapping Engine(SME) 기법을 사용하고 있다. 이를 위해 패턴과 프레임워크에 관련된 정보를 IMPORT 언어로 기술하도록 하였다. IMPORT 언어로 기술된 도메인 지식은 트리 형태로 재 구성된다. 매칭의 예는 다음과 같다.



(그림 2) 매칭 측정의 예
(Fig. 2) Analogical measure

트리로 표현하여 SME에 의해 까만 부분이 매치되는 부분으로 식별되면 트리의 레벨별로 일치되는 정도를 구한다. (1*1) + (2*2) + (3*3) + (4*3) + (5*1) = 31의 측정치를 얻을 수 있다. A1 지식 관점에서의 유사도는 31/41 = 0.66이 되고, A2의 관점에서는 31/81 = .37이 된다. 이를 평균한 0.51이 두 모델간의 유사도로 측정하는 방식을 취하고 있다.

본 연구에서는 SME와 ACME의 입력 조건을 만족하는 형태의 지식 표현이 아닌 객체 모델 자체를 입력물로 객체 모델이 재사용될 수있도록 지원한다. Maiden이 제안한 소프트웨어 산출물의 재사용을 위해 필요한 고려 사항들을 바탕으로 단위 객체 뿐 아니라 패턴과 객

체 행위 및 패턴 행위를 표현할 수 있는 표현 방법을 제안하였으며, 이들간의 유사도를 평가하는 방법을 제안하였다. Whitehurst과 달리 다양한 객체 관련성과 객체 행위를 고려한 유사도 평가 방법을 제안하였다.

3. Analogical Matching 기법

객체와 패턴을 컴포넌트로 라이브러리에 저장하기 위한 구조와 표현 기법을 정의하고, analogical matching을 기반으로 하여 라이브러리를 검색할 수 있는 기법을 정의하였다.

3.1 컴포넌트의 개념적 모델

재사용 대상으로서의 객체 모델은 객체 모델의 규모와 추상화 수준에 따라 각각 두 레벨로 구분했다. 재사용 단위의 규모에 따라서는 재사용 단위 중 가장 작은 각각의 객체와 보다 큰 규모의 재사용 대상인 패턴으로 구분하였다. 또한 표현하고자 하는 컴포넌트의 추상화 수준에 따라서 스키마(schema) 수준과 사례(case) 수준으로 구분하였다. 스키마란 개념에 대한 추상화로 정의되며 추상화된 모델, 추상화된 표현, 또는 템플리트(template)라고도 한다. 사례는 개념에 대한 어떤 구체적 실 예라고 정의할 수 있다.

본 논문에서는 어느 특정 방법론으로 구축한 객체 모델을 대상으로 하기보다는 일반적인 객체 모델의 재사용을 목표로 하였다. 일반적인 객체 모델로써 OMG (Object Management Group)에서 표준화하고 있는 객체 참조 모델을 기반으로 하였다[16]. OMG가 정의한 객체 모델 타입을 기반으로 하여 본 논문에서 표현하고자 하는 재사용 모델의 범위를 <표 2>와 같이 정의하였다.

〈표 2〉 OMG 참조 모델의 개념
〈Table 2〉 Concepts of OMG Reference Model

컴포넌트 / 속성	객체	패턴	
정적 특성	객체 타입(클래스) 속성 정보 오퍼레이션 정보	관련성 정보	상속관계(상세화관계) 집합관계 연결관계
동적 특성	이벤트 상태 전이 메시지	사용관계	
	전제조건		

3.2 모델의 라이브러리 표현 기법

재사용 대상에 대한 analogy를 평가한다는 것은 두 대상의 정적 특성과 동적 특성의 유사도를 평가하는 것이다. 객체 모델에 대한 OMG 표준화 참조 모델을 재구성하여 재사용 대상 별로 기술 양식을 정의하였다. 객체의 단일 구조에 대해서는 객체의 정적 구조를 시그네처(signature)로 정의하고 시그네처에 기술된 인터페이스를 기준으로 하여 동적인 특성을 객체 명세(specification)로 정의하였다. 패턴에 대해서는 패턴을 이루고 있는 객체들에 대한 관련성을 중심으로 패턴 시그네처와 패턴 명세로 정의하였다.

3.2.1 객체 시그네처

객체 모델을 구성하는 최소 단위인 객체를 재사용 라이브러리에 저장하고 이에 대해 analogical matching을 수행하기 위해서는 (그림 3)과 같은 특성으로 객체를 표현한다.

```
Object   (ObjectID, ObjectName, set of Attributes, set-of-Operations, ObjectKind)
ObjectKind   resource | process | history | logic | role | interaction
Attribute = (AttributeName, AttributeType, AttributeKey, AttributeKind, AttributeStructure)
AttributeType   data type of attribute
AttributeKey = identifier | non-identifier
AttributeKind   basic | derived | acquired
AttributeStructure   simple | composite
Operation = (OperationName, ResultType, set of Arguments)
Argument = (ArgumentName, ArgumentType, ArgumentStructure)
ResultType   ArgumentType = data type
ArgumentStructure = simple | composite | object
```

(그림 3) 객체 시그네처 표현
(Fig. 3) Representation of Object Signature

ObjectID는 재사용 라이브러리에 등록된 객체에 대한 고유한 식별자로서 라이브러리 내에서 식별자로 생성되는 번호이며, ObjectName은 객체 모델에서 부여한 객체의 이름이다. set of Attributes는 객체가 갖는 정적 속성들의 집합으로 속성들의 이름만 기술한다. set-of-Operations는 객체의 서비스를 나타내는 인터페이스들의 집합이다. ObjectKind는 객체의 유형을 정의하는 것으로 객체가 미리 정의한 유형들 중 어느 부류에 속하는지를 표현한 것으로 설계 라이브러리에 영역에 대한 정보나 일반적 정보를 기술하는 부분에 Is-a lattice로 표현되며, 계속적으로 추가할 수 있다.

3.2.2 객체 명세

객체 명세 부분에서는 한 객체가 가지고 있는 상태와 행위 개념을 표현하므로써 객체의 구문적 특성에다가 인터페이스의 의미를 추가하여 정의한다. 이는 객체에 대한 동적 모델링에 해당하는 부분이다.

(1) 상태 도메인에 대한 표현

객체의 행위 중 어느 한 단계를 나타내는 상태는 객체에 대해 구문적으로 정의된 속성들의 조합으로 표현할 수 있으며, (그림 4)와 같이 정의한다.

```
State          (set-of-StateDomains, set of Invariants,
                set of StateConstraints)
StateDomain    = AbstractState | Predicate
AbstractState  = abstract concept of state
Predicate      = concrete predicate of state
Invariant      = always predicate
StateConstraint = (AttributeName, Relation,
                   SourceAttribute)
```

(그림 4) 상태 도메인 표현
(Fig. 4) Representation of State Domain

상태 도메인은 어플리케이션에서 객체가 갖는 의미를 정의하는 중요한 요소이다. 동일한 객체 이름과 동일한 오퍼레이션을 갖는 객체라 하더라도 어플리케이션에서 요구되는 필요성에 따라 서로 다른 상태 도메인을 가질 수 있다.

동적 모델에서 표현되는 상태는 대부분 여러 속성 값들의 조합으로 정의할 수 있는 추상적 개념으로 표현하며, 이를 AbstractState라고 한다. 또한 보다 정확한 의미를 전달할 수 있는 구체적인 조건식(predicate)으로 정의할 수도 있다. 객체의 본질적 특성을 정의하는 데 필요한 Invariant는 객체가 어떤 행위를 수행하더라도 만족해야 하는 특성으로 always라는 예약어와 함께 만족해야 하는 조건으로 표현한다.

Constraint는 Invariant와 비슷한 것으로 한 속성과 다른 속성간의 상대적 관련성을 정의한 것으로 한 속성이 다른 속성에 대해 어떤 특성을 가지고 있는지를 관계식으로 정의하여 표현한다.

(2) 행위에 대한 표현

객체의 행위는 객체의 상태와 함께 객체의 중요한 동적 의미를 제공하며, 어떤 객체가 이벤트에 의해 한 상태로부터 다른 상태로 전이되기까지를 의미한다. 이것을 (그림 5)와 같이 정의하였다.

```
Behavior = (EventName, Pre-State, Post-State,
            set-of-UpdateAttributes,
            set-of-BehaviorConstraints)
Pre-State = StateDomain
Post-State = StateDomain
UpdateAttribute = AttributeName
BehaviorConstraint = (EventName, Relation,
                      EventName)
```

(그림 5) 행위 표현
(Fig. 5) Representation of Behavior

상태와 정의된 조건에 부합하는 메시지는 해당 객체로 하여금 정해진 행위를 수행하여 자신의 상태를 변화시키도록 한다. 행위는 행위를 유발하는 이벤트의 EventName과 행위를 수행하기 위해 만족해야 하는 선행조건인 Pre-State와 행위를 수행한 후 형성되는 Post-State로 정의한다. 추가하여 자신에게 정의된 속성 가운데 행위 도중 값의 변화가 발생하는 속성들의 이름을 UpdateAttribute로 표현한다. Constraint는 행위들간의 상관 관계에 대한 표현으로 이벤트들간의 발생 순서에 관한 조건을 기술한다.

3.2.3 패턴 시그네쳐

객체보다 큰 규모의 재사용 단위인 패턴은 객체들간의 관련성 구조를 포함하고 있다[5]. 패턴의 구문적 표현은 패턴 시그네쳐로 정의하고 의미적 표현은 패턴 명세로 정의하였다. 패턴 시그네쳐에는 해결해야 할 문제에 본질적으로 필요한 클래스와 그들간의 구조적 정보

의 정의한다. 패턴 시그네쳐 표현은 (그림 6)과 같이 패턴을 구성하는 객체들간의 관련성 정보를 중심으로 표현한다.

```
Pattern = set of Relationships
Relationship = Inherit | Aggregate | Associate
Inherit = (SuperClassName, set-of SubClasses,
           Discriminator)
SuperClassName = SubClass = ObjectName in
                 object signature
Discriminator = abstract concept
Aggregate = (WholeClassName, Multiplicity,
             set-of-PartClasses)
PartClass = (PartClassName, Multiplicity)
WholeClassName = PartClassName = ObjectName
                 in object signature
Associate = (AssociateName, (LhsClassName,
             Multiplicity, role),
             (RhsClassName, Multiplicity, role))
LhsClassName = RhsClassName = ObjectName in
               object signature
Multiplicity = 0 | 1 | n | * | Range
Range = integer '..' integer
n : integer
```

(그림 6) 패턴 시그네쳐 표현
(Fig. 6) Representation of Pattern Signature

패턴을 구성하는 객체들간의 관련성은 상속 관계, 집합 관계, 연결 관계로 정의한다. 상속 관계의 경우는 상위 클래스 이름과 하위 클래스들의 이름으로 표현하며, 이러한 상속 관계를 구성하게 된 추상적 관점을 판별자(discriminator)로 표현한다.

집합 관계의 경우는 전체 클래스와 구성 요소 클래스들간의 다중성 관계로 표현하며, 연결 관계의 경우는 두 객체 사이의 연결 관계명과 연결 관계를 구성하는 두 객체에 관한 정보를 표현한다. 연결 관계 다이아그램을 중심으로 왼쪽 객체와 오른쪽 객체를 표현하며 이들 각각에 대해 다중성 정보와 연결 관계에서의 역할 정보를 기술한다.

3.2.4 패턴 명세

패턴 명세는 패턴이 가지고 있는 의미를 패턴을 구성하는 객체들간의 협력 관계로 표현한 것이며 (그림 7)과 같다. 패턴의 활용과 의미에 대한 정보를 표현하기 위해 주로 패턴을 구성하는 객체들간의 행위적 특성을 정의하는데 주력하였다. Trigger 관계는 패턴을 구성하는 객체들 사이의 메시지 송수신 관계를 정의

```
Trigger = (SourceObject, Operation, TargetObject,
           TargetOperation)
SourceObject = TargetObject = ObjectName in object
               signature
Operation ∈ Operation(SourceObject)
TargetOperation ∈ Operation(TargetObject)
```

(그림 7) 패턴 명세 표현
(Fig. 7) Representation of Pattern Specification

3.2.5 설계 라이브러리

모델의 재사용은 코드의 재사용과 같이 개념이나 정보가 동일한 용어나 동일한 구조로 표현되어 있는 경우에만 재사용 가능하다고 판단할 수는 없다. 질의 객체의 이름이 컴포넌트 객체의 이름과 서로 다르다 하더라도 두 객체가 해당 문제에서 공유할 수 있는 개념을 갖는다면 두 객체는 서로 유사하다고 판단해야 하기 때문이다. 이와 같이 일치되는 개념뿐만 아니라 유사성의 관점에서도 매칭 시키려면 모델 컴포넌트 이외에도 부가적으로 도메인에 대한 일반적인 지식을 저장해야 한다. 이러한 일반적 지식은 객체 지향에서 사용되는 일반화 개념과 구성 관계를 적용하여 표현하며, 객체지향적 의미로 해석한다. 설계 라이브러리에 표현되는 개념을 (그림 8)과 같이 정의하였다.

```
BasicType (datatype)
Is-a (Kind, SubType, SuperType)
Kind = Object | Attribute | Operation | BasicType
Part-of (CompositeType, MemberType)
```

(그림 8) 설계 라이브러리 표현
(Fig. 8) Representation of Design Library

BasicType은 재사용 라이브러리에 표현할 기본 타입을 기술한 것으로 기본 타입으로 사용되는 자료형을 정의한다. Is-a는 SubType이 SuperType의 상세화된 의미라고 정의한 것이다. Kind는 일반화 관계의 대상을 정의하는 것으로 본 논문에서는 객체, 속성, 오퍼레이션, 기본 타입간의 일반화 관계로 한정한다. 일반화 관계는 상위 타입을 하위 타입으로 대체하거나 하위 타입을 상위 타입으로 대체하여 의미적 동질성을 파악하고자 할 때 활용된다. Part-of 관계는 CompositeType과 MemberType들의 관계를 표현한다.

3.3 Analogical Matching 함수

본 절에서는 대상들간의 analogy를 규명하는데 필요한 논리를 정립하고자 한다. 두 대상간의 analogy를 평가한다는 것은 두 대상의 정적 특성과 동적 특성의 유사도를 평가하는 것이다. 이를 위해 객체에 대한 매칭과 패턴에 대한 매칭을 각각 정의하였다.

객체의 정적 특성에 대해서는 인터페이스간의 일치도를 평가하는 인터페이스 매칭 함수를 정의하였으며, 객체의 동적 특성을 평가하기 위해서는 명세 매칭 함수를 정의하였다. 객체들간의 상호 작용을 정의한 패턴의 경우는 패턴을 구성하는 객체들간의 관련성 일치도를 평가하는 패턴 구조 매칭 함수와 패턴을 구성하는 요소들간의 상관 관계에 따른 행위 일치도를 평가하는 패턴 명세 매칭 함수를 정의하였다.

먼저 일반적인 analogical matching을 (정의 1)과 같이 정의한다.

(정의 1) 질의(Q)와 컴포넌트(C)간의 Analogical Matching 함수 AMatch(Q, C)

$amatch(Q_{signature}, C_{signature}) \vee amatch(Q_{specification}, C_{specification}) \vee amatch(Q_{pattern}, C_{pattern}) \vee amatch(Q_{framework}, C_{framework})$

이상은 질의 모델과 컴포넌트 모델간에 객체 수준의 시그네처가 유사하거나 명세가 유사하던지, 패턴 수준의 인터페이스가 유사하거나 패턴의 상호 작용이 유사하다면 두 대상간에 analogy가 존재하는 것으로 판단한 것이다. 질의와 컴포넌트 사이에 analogy가 존재한다는 것은 질의나 컴포넌트에 적용할 수 있는 일련의 변형 규칙(transformation rule)이 존재한다는 것을 의미한다. 이를 위해 필요한 기본 매칭 함수를 다음과 같이 정의한다.

(정의 2) 기본 매칭 함수

(1) Type Equality $T_1 =_e T_2$

① T_1과 T_2의 이름이 같을 경우

② T_1과 T_2의 구문적 성격으로 자료형이 같을 경우

③ 두 타입 $T_1(e_1,e_2,...e_n)$, $T_2(m_1,m_2,...,m_m)$가 모두 복합 구조인 경우

$\forall e_i, \exists m_j \cdot e_i =_e m_j,$ (단, $1 \le i \le n$, $1 \le j \le m$)

④ 하나의 타입 $T_1(e_1,e_2,...e_n)$이 복합 구조인 경우

$\exists e_i \cdot e_i =_e T_2$ (단, $1 \le i \le n$)

(2) Renaming $E =_r E'$

① $R(E) =_r E'$ or

② $Is\_a(E, E') \in DesignLibrary$

(3) Type Equivalence $T_1 =_e T_2$

① $T_1 =_r T_2$

② 두 타입 $T_1(e_1,e_2,...e_n)$, $T_2(m_1,m_2,...,m_m)$가 모두 복합 타입일 경우

$\forall e_i, \exists m_j \cdot e_i =_r m_j,$ (단, $1 \le i \le n$, $1 \le j \le m$)

③ 하나의 타입 $T_1(e_1,e_2,...e_n)$이 복합 타입인 경우

$\exists e_i \cdot e_i =_r T_2,$ (단, $1 \le i \le n$)

(4) Matching by reOrdering $O_1 =_o O_2$

$\forall e_i \in O_1, \exists m_j \in O_2 \cdot e_i =_e m_j,$

(단, $1 \le i \le n$, $1 \le j \le m$)

O : Parameter-Set|Attribute-Set|Operation-Set

e_i, m_j : ParameterName|AttributeName|OperationName

(정의 2)의 (1)은 매칭 하고자 하는 두 대상의 타입이 일치하는지를 판단하기 위한 기준을 정의한 것이며, (2)는 매칭 하고자 하는 두 대상의 이름이나 자료형이 같지 않더라도 동질성을 인정할 수 있는가를 확인한다. 또한 (정의 2)의 (3)에서는 매칭 하고자 하는 두 대상의 타입이 완전히 일치하지는 않더라도 의미적 동질성을 가질 수 있는 기준을 정의했으며, (4)에서는 매칭 하고자 하는 대상이 파라미터, 속성 이름, 오퍼레이션 이름인 경우, 이들이 모두 집합의 개념으로 표현되기 때문에 기술된 순서와 상관없이 매칭 시킬 수 있다는 것을 정의했다.

시그네처에 표현되는 구문적 특성을 변형하기 위해 필요한 함수들 외에도 모델의 목적과 의도를 파악하여 매칭하기 위해서는 명세를 바탕으로 한 의미론적 매칭이 필요하다.

의미론적 매칭을 위해 사용한 기본적 개념은 동형사상(homomorphism)과 이형사상(isomorphism) 이론이다[8]. 먼저 일반적인 동형사상과 합동사상을 정의하면 (정의 3)과 같다.

(정의 3) Order sorted Σ-동형사상, Σ-이형사상

$\Sigma=(S,\le,F)$: 시그네처, $A,B \in Alg(\Sigma)$.

Σ-동형사상 $\phi : A \to B$은 다음의 조건을 만족

아는 $\{\phi_s:A_s \to B_{s'}\}_{s\in S}$ 함수의 집합을 정의

조건 ① $\phi_s(f^A_{s1\ldots sns}(a_1,\ldots,a_n)) = f^B_{s'1\ldots s'ns}(\phi_{s1}(a_1), \ldots, \phi_{sn}(a_n))$
for all $f:(s_1,\ldots,s_n) \to s\in F$, $a_i\in A_{si}$, $i=1,\ldots,n$.

조건 ② $s\le t$ implies $\phi_s(a)=\phi_t(a)$ for all $a\in A_s$. [1]

위의 (정의 3)에서 ①은 동형사상을 정의한 것으로 A에 정의된 매핑에 대해 일련의 매핑 함수 f에 의해 B에서 대응되는 매핑을 발견할 수 있음을 의미하고, ②는 함동사상을 정의한 것으로 A에서 B로 대응될 수 있을 뿐만 아니라 B에서 A로도 대응될 수 있음을 의미한다.

3.3.1 객체 시그네처 매칭

객체의 유형이 완전히 일치하거나 설계 라이브러리에 정의된 도메인 정보를 바탕으로 동질의 유형을 갖는 것으로 판단되면 유사하다고 간주할 수 있다. 또한 정의한 속성들간에 일치성이나 동질성이 발견되거나 오퍼레이션들간의 특성이 동질이면 두 객체는 유사한 것으로 간주한다.

(정의 4) 객체 질의 대 객체 컴포넌트간의 완전 매칭 함수 $\text{Exact\_amatch}(OQ_{interface}, OC_{interface})$

① $(ObjectKind(OQ) =_e ObjectKind(OC) \vee ObjectKind(OQ) =_q ObjectKind(OC))$ 또는

② $(\forall attribute_q \in set\text{-}of\text{-}Attributes(OQ), \exists attribute_c \in set\text{-}of\text{-}Attributes(OC) \cdot attribute_q =_e attribute_c \vee attribute_q =_q attribute_c)$ 또는

③ $(\forall operation_q \in set\text{-}of\text{-}Operations(OQ), \exists operation_c \in set\text{-}of\text{-}Operations(OC) \cdot operation_q =_e operation_c \vee operation_q =_q operation_c)$

(정의 4)는 질의에 표현된 모든 특성이 컴포넌트에도 표현되어 있는 경우를 식별하기 위한 것이다. 이 경우에는 질의에 표현되지 않은 것이 컴포넌트에 존재할 수 있으며, 컴포넌트에 추가적으로 표현된 사항을 바탕으로 질의를 재정의할 수 있다.

(정의 5) 객체 질의 대 객체 컴포넌트간의 부분 매칭 함수 $\text{Partial\_amatch}(OQ_{interface}, OC_{interface})$

① $(\forall attribute_c \in set\text{-}of\text{-}Attributes(OC), \exists attribute_q \in set\text{-}of\text{-}Attributes(OQ) \cdot attribute_q =_e attribute_c \vee attribute_q =_q attribute_c)$ 또는

② $(\forall operation_c \in set\text{-}of\text{-}Operations(OC), \exists operation_q \in set\text{-}of\text{-}Operations(OQ) \cdot operation_q =_e operation_c \vee operation_q =_q operation_c)$

(정의 5)의 경우는 컴포넌트에 표현된 모든 것이 질의의 요소에 대응될 수 있으나, 질의에 표현된 요소들 중 일부는 컴포넌트에 표현되어 있지 않을 수도 있는 경우이다. 그러나 이런 경우 역시 컴포넌트를 통해 질의를 재정의 하거나 컴포넌트를 상속받아 질의를 재구성할 수 있으므로 유사한 것으로 판단한다. 이를 부분 매칭 함수로 정의하였다.

(정의 6) 객체 질의 대 패턴 컴포넌트간의 부분 매칭 함수 $\text{Partial\_amatch}(OQ_{interface}, PC_{interface})$
$(\exists attribute_q \in set\text{-}of\text{-}Attributes(OQ), \exists object_c \in set\text{-}of\text{-}Objects(PC) \cdot attribute_q =_e Object_c \vee attribute_q =_q Object_c)$

(정의 6)은 객체로 작성된 질의와 재사용 라이브러리에 저장된 패턴간의 구조적 유사성을 파악하는 부분 매칭 함수이다. 이는 어떤 객체에서 속성으로 표현된 것이 다른 모델에서는 독립된 객체로 표현될 수 있는 경우를 식별하기 위한 함수이다.

3.3.2 객체 명세 매칭

객체들간의 행위적 유사성을 판단하기 위한 객체 명세 매칭은 객체의 상태 도메인에 대한 일치 성도와 행위의 일치성을 판단하는 것이다. 또한 객체의 속성에 대한 제약 조건이 존재할 경우 이 조건에 대한 일치성도 고려해야 한다.

(정의 7) 행위 동질성에 의한 행위 매칭 함수 $\text{amatch}(OQ_{behavior}, OC_{behavior})$
if $Pre\text{-}State(operation_q) =_e Pre\text{-}State(operation_c)$ or $Pre\text{-}State(operation_q) =_q Pre\text{-}State(operation_c)$

then Post-State(operation$_q$) $\equiv_e$ Post-State(operation$_c$) or Post-State (operation$_q$) $\equiv_q$ Post-State(operation$_c$)

(정의 7)은 서로 유사한 상태에서 유사한 오퍼레이션에 대해 반응하는 방식이 유사함을 의미한다. 이는 대부분의 객체 모델이 추상화된 상태 개념(Full, Waiting for something 등)으로 표현되지만, 구체적 조건식(x.time < 100 ∧ y.interval > 10)으로 표현되는 경우도 있다. 이와 같이 구체적 조건식으로 표현되는 경우는 기존의 명세 재사용에서 이루어진 기법을 적용할 수 있다. 기존의 명세의 재사용에서는 함수의 선행 조건과 후행 조건간의 implication 원칙이 성립되면 기존의 명세를 재사용할 수 있는 것으로 정의하였다. 객체의 단위 행위는 하나의 함수로 대응 시킬 수 있고 한 함수에 대한 모델을 재사용하는 관점에서 (정의 7)을 적용하였다.

그러나 객체 자체의 행위는 이벤트에 의한 상태 전이의 과정으로 표현할 수 있고 이는 단위 함수들이 모여 객체의 전체 동적 특성을 표현한다. 이와 같은 객체의 동적 특성은 (정의 7)로 판단하기에 제약성을 갖는다. 객체의 전체 동적 특성간의 매칭은 (정의 8)로 정의하였다.

(정의 8) Behavior Graph BG(V, E)

$V = \{v_i \mid v_i \text{ is AbstractState}\}$

$E = \{(u, v, l) \mid (u, v \in V) \wedge (l \text{ is EventName})\}$

객체의 동적 특성을 나타내는 행위는 방향성 그래프로 표현할 수 있다. (정의 8)에서 행위에 표현되는 상태는 그래프의 노드로 표현하고, 한 상태로부터 다른 상태로의 전이는 방향성 연결선(edge)으로 표현한다. 이벤트 이름은 연결선의 레이블로 표현된다.

(정의 9) 객체 명세의 완전 매칭 함수 Exact_amatch (BGQ, BGC)

$\forall (u,v,l) \in BGQ \cdot \exists (f(u), f(v), f(l)) \in BGC,$ (단, $f : E \rightarrow E^*$)

// BGQ(V, E) : 질의로 작성된 객체의 행위 그래프

BGC(V^*, E^*) : 컴포넌트로 저장된 객체의 행위 그래프 //

객체의 행위를 그래프로 표현함에 따라 객체 행위에 대한 analogical matching은 그래프의 동형 사상과 합동 사상에 대한 개념을 적용할 수 있다.

(정의 9)에서 정의한 객체 명세의 완전 매칭은 객체의 행위 유형이 완전히 일치하는 경우로 질의에 정의된 각 상태 전이 규칙이 컴포넌트의 상태 전이 규칙과 대응될 수 있다는 것을 의미한다. 그러나 이러한 이형사상에 의한 매칭은 너무 제한적이기 때문에 (정의 10)에서는 동형사상에 의한 부분 매칭을 제안한다.

(정의 10) 객체 명세의 부분 매칭 함수 Partial_amatch(BGQ, BGC)

① $\exists (u,v,l) \in BGQ \cdot \exists (f(u), f(v), f(l)) \in BGC,$ (단, $f : E \rightarrow E^*$)

② BGQ'에 대해 Exact_amatch(BGQ', BGC)가 존재

③ BGC'에 대해 Exact_amatch(BGQ, BGC')가 존재

// BGQ'(V',E') : BGQ를 확장한 그래프

$V' = V \cup \{v_i\}$,

$E' = E \cup \{(v_j,v_i,l'), (v_i,v_k,l'')\}$

(단, $v_j,v_k \in V$, $v_i \notin V$, $(v_j,v_k,l) \in E$, $(v_j,v_i,l'),(v_i,v_k,l'') \notin E'$, l',l'' : 새로 정의할 수 있는 이벤트)

BGC'(V',E') : BGC를 확장한 그래프

$V' = V \cup \{v_i\}$

$E' = E \cup \{(v_j,v_i,l'), (v_i,v_k,l'')\}$

(단, $v_j,v_k \in V$, $v_i \notin V$, $(v_j,v_k,l) \in E$, $(v_j,v_i,l'),(v_i,v_k,l'') \notin E', l',l''$: 새로 정의할 수 있는 이벤트) //

(정의 10)의 ①은 기본적으로 질의에 존재하는 행위 유형중 하나라도 컴포넌트에서 발견할 수 있다면 재사용 후보 모델로 간주할 수 있음을 의미한다. 이를 바탕으로 (정의 10)의 ②와 ③에서는 질의의 행위나 컴포넌트의 행위에 새로운 상태(vi)를 추가하므로써 질의의 행위와 컴포넌트의 행위간에 완전 일치가 이루어진다면 둘 간에는 부분 매칭이 성립하는 것으로 간주한다. 이는 새로운 상태를 추가하고 새로운 상태 전이 규칙을 정의하여 두 행위 그래프간에 이형사상을 정의할 수 있는가를 판단하는 것이다. 이로써 서로 다른

상대 도메인의 경우나 전의 규칙이 완전히 일치하지 않는 경우도 행위의 구조가 유사하면 재사용 후보로 식별할 수 있다.

(정의 11) 행위 순응에 의한 행위 매칭 함수 amatch ($Q_{behavior}$, $C_{behavior}$)

① Pre-State(Q) ⇒ Pre-State(C)

② (Pre-State(Q) ⇒ Post-State(Q)) ⇔ Post-State(C)

3.3.3 패턴 시그네쳐 매칭

패턴은 하나의 클래스로만 구성된 것이 아니라 클래스들간의 관련성을 포함하고 있기 때문에 패턴에 대한 매칭을 수행하는 것은 보다 복잡하다. 이를테면 어떤 모델에서는 하나의 클래스에 속성과 오퍼레이션으로 정의된 것이 다른 모델에서는 서로 다른 클래스로 구분되고 이들간의 관련성으로 정의할 수도 있기 때문이다. 이런 경우라면 두 모델을 서로 유사한 것으로 판단해야 한다. 이를 위해서 먼저 패턴을 구성하는 관련성에 대한 의미를 바탕으로 매칭을 정의해야 한다. 이는 질의 객체 모델에 정의된 관련성이 컴포넌트 객체 모델에는 정의되어 있지 않지만, 상속이나 집합 관계를 통해 하나의 클래스에 모두 다 정의되어 있을 수도 있기 때문이다.

(정의 12) 상속 관계 확장을 위한 기본 함수 Extend (set-of-InheritedClasses)

```
for each Inherit
  ExtendClass = (Name, set-of-Attributes,
    set-of-Operations)
  Name = name of SuperClass
  Attribute = set-of-Attributes(SuperClass)
    ∪ set-of-Attributes(SubClasses)
  Operation = set-of-Operations(SuperClass)
    ∪ set-of-Operations(SubClasses)
  Pattern = (Pattern - InheritedClasses)
    ∪ ExtendClass
  if Relationship(InheritedClass, Objectm) =
    Aggregate or Associate then Pattern =
      (Pattern - Relationship(InheritedClass,
        Objectm)) ∪ Relationship(Extend-
        Class, Objectm)
end for
// InheritedClass = {(Objecti, Objectj) ∈
      Pattern | Relationship(Objecti,
      Objectj) = Inherit}    //
```

(정의 12)는 상속 관계를 포함하고 있는 패턴을 상속 관계가 없는 하나의 객체로 해석하기 위한 것이다. Extend 함수에 의해 상속 관계를 갖는 클래스들에 정의된 속성과 오퍼레이션을 모두 합집합하여 새로운 하나의 클래스로 확장한 후, 기존의 상속 관계로 표현된 클래스들을 패턴에서 제거하고 새로 확장한 클래스를 패턴에 추가하여 매칭을 시도할 수 있다. 또한 상속 관계에 정의된 클래스들 중 어느 한 클래스와 집합 관계나 연결 관계를 맺고 있는 다른 클래스가 패턴에 존재한다면 새롭게 정의된 클래스와 관련성을 갖도록 변경한다.

(정의 13)은 집합 관계를 포함하고 있는 패턴을 집합 관계가 없는 하나의 객체로 간주하기 위한 것이다.

(정의 13) 집합 관계 평면화를 위한 기본 함수 Flat (set-of-AggregatedClasses)

```
for each Aggregate
  FlatClass = (Name, set-of-Attributes,
      set-of-Operations)
  Name = name of WholeClass
  Attribute = set-of-Attributes(WholeClass)
    ∪ set-of-Attributes(PartClasses)
  Operation = set-of-Operations(WholeClass)
    ∪ set-of-Operations(PartClasses)
  Pattern = (Pattern - AggregatedClasses)
    ∪ FlatClass
  if Relationship(AggregatedClass, Objectm)
      = Inherit or Associate
      then Pattern = (Pattern - Rela-
        tionship(AggregatedClass, Ob-
        jectm)) ∪ Relationship(FlatClass,
        Objectm)
end for
// AggregatedClass = {(Objecti, Objectj)
      ∈ Pattern | Relationship(Objecti,
      Objectj) = Aggregate}    //
```

(정의 12)와 (정의 13)에서는 객체지향 개념의 관련성에 대한 의미론을 바탕으로 질의 패턴과 재사용 라이브러리에 저장된 컴포넌트 패턴간의 매칭을 판단하기 위한 기본 함수를 정의하였다.

이와 더불어 (정의 3)의 동형사상과 이형사상에 대한 정의를 패턴의 매칭에도 적용할 수 있다. 패턴은 기본적으로 동적 모델과 동일하게 그래프로 표현될 수 있기 때문이다.

패턴에 대한 그래프 표현은 (정의 14)와 같다. 그래프의 정점은 객체를 나타내며, 연결선의 꼬리 쪽은 상속 관계의 SuperClass, 집합 관계의 WholeClass 또는 연결 관계의 LhsClass를 나타내고, 머리 쪽은 상속 관계의 SubClass, 집합 관계의 PartClass 또는 연결 관계의 RhsClass를 나타낸다. 또한 관련성은 연결선의 레이블로 표현한다.

(정의 14) Pattern Graph PG(V, E)

V = {x | x is object}

E = {(u, v, r) | (u is SuperClass | Whole Class | LhsClass) ∧ (v is SubClass | PartClass | RhsClass) ∧ (r is Inherit | Aggregate | Associate)}

(정의 15) 패턴의 구조에 대한 완전 매칭 함수 Exact_amatch(PGQ, PGC)

for each element in PGQ, PGC

∀r ∈ E, ∃transformation function h from V onto V′ · (u,v,r) ∈ E iff (h(u), h(v), r)) ∈ E′ and

∀r ∈ E′, ∃transformation function h from V′ onto V · (u,v,r) ∈ E iff (h(u), h(v), r)) ∈ E′

end for

// PGQ = {Q(V,E), QE(V,E), QF(V,E)}

PGC = {C(V′,E′),CE(V′,E′),CF(V′,E′)}

Q(V,E) : 질의 그래프

QE(V,E) : Extend 함수를 적용한 질의 그래프

QF(V,E) : Flat 함수를 적용한 질의 그래프

C(V′,E′) : 컴포넌트 그래프

CE(V′,E′) : Extend 함수를 적용한 컴포넌트 그래프

CF(V′,E′) : Flat 함수를 적용한 컴포넌트 그래프 //

(정의 15)는 질의에 정의된 모든 객체와 관련성이 컴포넌트 패턴에 대응되고, 반대로 컴포넌트에 표현된 모든 객체와 관련성이 질의 패턴에 대응되는 경우로 패턴의 완전한 매칭이 이루어지는 경우를 정의한 것이다. 그러나 이는 질의로 작성된 패턴이 완전하게 작성된 경우에만 가능하므로 너무 제약적이다.

(정의 16) 패턴의 구조에 대한 부분 매칭 함수 Partial_amatch(PGQ, PGC)

for each element in PGQ, PGC

∃r ∈ E, ∃transformation function h from V onto V′ · (u,v,r) ∈ E iff (h(u), h(v), r)) ∈ E′

end for

(정의 16)은 질의에 정의된 모든 객체와 관련성 중 부분적으로 컴포넌트 패턴에 대응되는 경우를 정의한 것이다. 동형사상의 정의에 따라 매칭 하면 이형사상의 정의에 의한 완전 매칭보다 많은 패턴을 재사용 후보로 검색할 수 있다.

3.3.4 패턴 명세 매칭

패턴 명세 매칭의 경우는 객체들간의 관련성 구조가 정립된 상황에서 관련성에 따른 객체들간의 상호 작용 방식이 일치하는지를 판단하는 것이다.

(정의 17) Analogical Pattern Matching 함수 amatch ($Q_{framework}$, $C_{framework}$)

(ObjectID($Q_{trigger}$) $\equiv_o$ ObjectID($C_{trigger}$) ∨ OperationName($Q_{trigger}$) $\equiv_o$ OperationName($C_{trigger}$)) ⇒

(TargetObjectID($Q_{trigger}$) $\equiv_o$ TargetObjectID($C_{trigger}$) ∨ TargetOperation($Q_{trigger}$) $\equiv_o$ TargetOperation($C_{trigger}$))

(정의 17)은 패턴에서 객체들간의 상호 작용 방식이 일치하는 가를 판단하기 위한 것으로 질의 패턴과 컴포넌트 패턴에서 trigger 객체가 일치하거나 trigger 오퍼

패턴의 조작의 trigger를 받는 객체의 trigger를 받는 오퍼레이션이 동질성을 갖게 된다는 것을 의미한다.

패턴 명세의 경우도 패턴 시그내쳐 매칭의 경우와 마찬가지로 패턴을 그래프로 변환한 후 그래프 사상에 의해 매칭을 판단한다. (정의 18)은 패턴 명세를 그래프로 정의한 것이다. 여기에서 그래프의 정점은 객체를 나타내는데, 연결선의 꼬리 쪽은 ClientObject를 나타내고, 머리 쪽은 ServerObject를 나타낸다. 또한 Request는 연결선의 레이블로 표현한다.

(정의 18) Pattern Specification Graph PSG(V, E)

$V = \{x \mid x \text{ is object}\}$

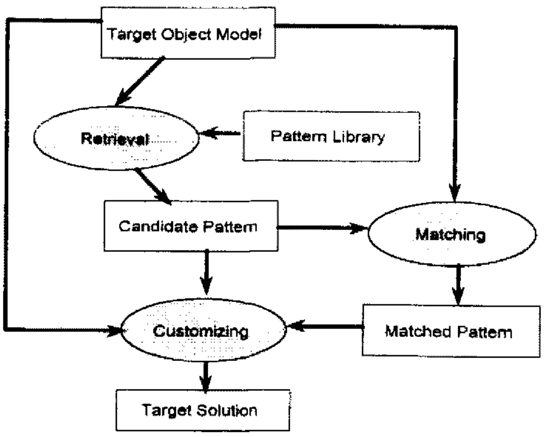
$E = \{(u, v, r) \mid (u \text{ is ClientObject}) \wedge (v \text{ is ServerObject}) \wedge (r \text{ is Request})\}$

4. 모델의 재사용 환경 구축 및 평가

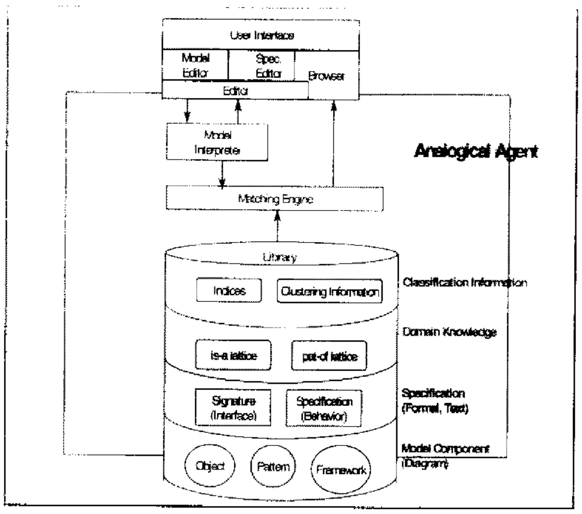
analogical matching에 의해 모델의 재사용을 지원할 수 있도록 정의된 라이브러리 구조와 매칭 함수를 적용하여 질의를 만족하는 컴포넌트를 찾을 수 있도록 검색 환경을 구축하고 평가하였다.

4.1 검색 시스템을 위한 기본 설계

검색은 모델 작성 과정에서 초기에 작성한 객체 단위에서부터 점차적으로 객체들과의 관련성을 갖는 패턴으로 확대되어 간다. 따라서 검색 과정은 완전한 모델 뿐만 아니라 부분적 객체 모델도 질의로 처리할 수 있어야 한다. 재사용 라이브러리를 검색하는 개략적인 시나리오는 (그림 9)와 같다.



(그림 9) 재사용 시스템의 전체 흐름도
(Fig. 9) Flow Diagram of Reuse System



(그림 10) Analogical Agent의 구조
(Fig. 10) Architecture of Analogical Agent

사용자의 모델 작성 과정과 라이브러리의 검색 과정을 함께 지속적으로 모니터링 할 수 있는 analogical agent를 개발하였다. 이 에이전트에 대한 기본 구조는 (그림 10)과 같다.

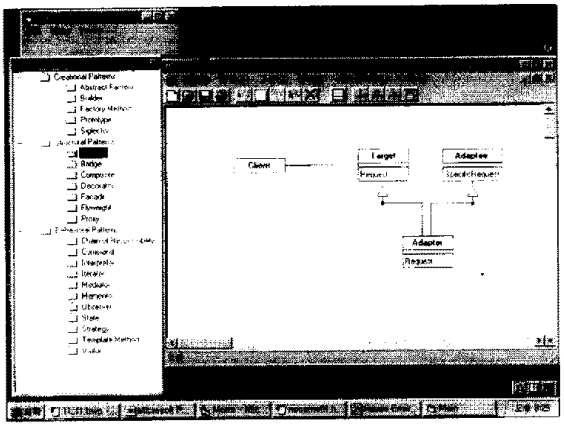
4.2 구현 예

본 시스템은 Visual Basic 개발 환경으로 개발하였으며, 크게 모델 작성기와 모델 검색기로 구성되어 있다. 모델 검색기는 검색된 하나의 모델에 대한 사례를 살펴 볼 수 있도록 모델 이해 지원기를 포함하고 있으며, 라이브러리를 관리하는 부가적인 서브 시스템을 포함하고 있다.

4.2.1 라이브러리

라이브러리는 컴포넌트를 저장하고 있는 라이브러리와 도메인에 대한 개념을 제공하는 설계 정보 라이브러리로 구분하여 관리된다. 컴포넌트 라이브러리에는 객체와 패턴들이 저장되어 있지만, 실제로 패턴이란 라이브러리에 저장된 객체들 간의 관련성을 정의한 객체 보다 한 추상화 단위일 뿐이다. 따라서 컴포넌트에 대한 이해를 돕기 위해 각 객체 모델이나 패턴에 대한 실제 적용 예를 제공하였다. (그림 11)은 라이브러리에 저장된 패턴의 종류에 대한 계층도와 그 중 Adapter에 대한 패턴 컴포넌트를 보이 주고 있다.

이 패턴 계층도는 Gamma가 제안한 패턴 구조[6]를 따르고 있다. 현재 라이브러리는 Gamma의 설계 패턴[24]과 Fowler가 제안한 분석 패턴들로 구성되어 있다[5].



(그림 11) 라이브러리의 패턴 계층도
(Fig. 11) Hierarchy of Patterns in Library

4.2.2 모델 작성기

모델 작성기는 사용자가 클래스 다이아그램과 관련성 다이아그램을 그려서 질의로 사용할 수 있도록 지원하는 도구이다. 기본적으로는 특정 모델링 방법론에 종속적이지 않아도 되지만, 기준이 되는 다이아그램 표기법이 필요했다. 이를 위해 가장 널리 사용되는 객체 모델링 다이아그램인 OMT(Object Modeling Tool)의 형태를 갖추도록 하였다[13].

4.2.3 모델 검색기

모델 검색기는 작성한 질의를 바탕으로 하여 라이브러리를 검색하고 가장 유사한 후보 컴포넌트들을 검색하는 서브 시스템이다. 완전 매칭 수준의 검색과 부분 매칭 수준의 검색을 지원하며, 객체 단위의 검색과 패턴 단위의 검색을 지정할 수도 있다. 객체 단위의 검색으로 지정하면 하나의 객체를 작성하는 과정에서 지속적으로 라이브러리를 검색하여 후보 컴포넌트들을 제공하게 되고, 패턴 단위로 지정하면 패턴이 완성된 후 검색을 선택할 때 검색된다.

4.2.4 이해 지원기

이해 지원 시스템은 검색된 컴포넌트 모델에 대한 구체적인 내용을 확인할 수 있도록 클래스를 구성하는 관련성 정보, 클래스에 대한 자세한 속성 정보, 오퍼레이션에 대한 구체적인 정보들을 제공한다.

4.3 평가

본 논문에서 제안한 재사용 기법의 효율성을 검증하기 위하여 검색 환경에서 널리 쓰이는 평가 기법인 재현율(recall)과 정확성(precision)을 기준으로 평가하였다.

4.3.1 재현율과 정확도에 의한 평가

재현율은 관련 있는 컴포넌트를 검색에서 배제시킬 수 있는 능력에 관한 평가 기준으로, 검색된 컴포넌트들 중 실제 질의에 부합되는 컴포넌트의 비율로 정의된다. 정확도란 관련된 컴포넌트의 검색 능력에 관한 평가로서, 실제 라이브러리에 저장되어 있는 질의를 만족하는 컴포넌트들 가운데 검색된 컴포넌트의 비율로 정의된다[14].

$$\text{재현율} = \frac{W}{F} \qquad \text{(식 1)}$$

$$\text{정확도} = \frac{W}{R} \qquad \text{(식 2)}$$

객체 단위의 검색과 패턴 단위의 검색 각각에 대해 완전 매칭과 부분 매칭을 (식 1)과 (식 2)를 이용하여 평가한 결과는 <표 3>과 같다. 본 논문에서 실험 대상으로 한 영역은 기존에 패턴 책에서 제안한 다양한 설계 패턴과 분석 패턴을 모델 라이브러리로 구축하여 평가 하였다.

〈표 3〉 재현율, 정확도에 따른 평가 결과
〈Table 3〉 Recall, Precision Measure

(a) 재현율에 대한 평가 결과

	객체 검색	패턴 검색
완전 검색	56.3	44.2
부분 검색	75.8	60.3

(b) 정확도에 따른 평가 결과

	객체 검색	패턴 검색
완전 검색	78.6	67.2
부분 검색	66.3	58.5

재현율과 정확도에 의한 평가 기준을 본 논문에서 구축한 시스템에 적용한 결과 평균적인 기준[14]을 만족한다는 것을 알 수 있었다. 정확도와 검색 범위에 있어서는 객체 검색의 경우가 패턴 검색의 경우보다 좋다는 것을 알 수 있었다. 객체는 매칭 대상이 하나이므로 매칭 결과가 설계자의 의도에 따라 크게 좌우되지 않지만 패턴의 경우는 동일한 관련성을 정의하더라고 그 관련성을 통해 객체들간에 어떠한 상호 작용

의 어휘나며, 어떤의 행위가 상황의 ... 하는 것이 전적으로 설계자에게 달려 있기 때문이다. 이것은 패턴의 시그네쳐 매칭에서 동형사상을 적용하거나 패턴 명세를 기반으로 한 매칭을 하더라도 완전히 해결할 수 있는 부분이 아니다.

평가 결과에서도 패턴 매칭의 경우는 정확도가 떨어진다는 것을 알 수 있다. 이는 패턴 명세의 매칭에서 패턴의 구조를 중심으로 매칭 함수를 적용하였기 때문으로, 패턴을 구성하는 각 클래스의 특성이 해결하고자 하는 문제와 다를 수 있기 때문으로 생각된다. 그러나 부분 매칭을 통해서는 패턴의 경우도 만족할 만한 수준의 재현율을 얻을 수 있으므로, 이를 바탕으로 사용자의 수정을 거치면 문제 영역에서 요구되는 모델을 완성할 수 있다.

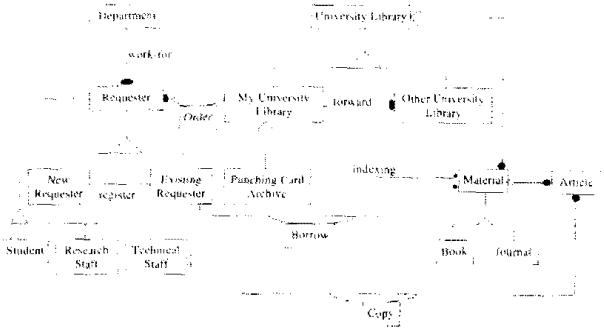
4.3.2 다른 분류 기법과의 비교

매칭 기법에 의한 모델과 패턴의 검색은 기존의 검색 시스템에서 사용하던 Facet 방법이나 계층 분류와 비교해 볼 때, 다음과 같은 장점을 갖는다. Facet 기법이나 계층 분류 기법의 경우는 라이브러리 개발자가 미리 모델에 대한 분류 기준을 갖고 Facet과 Facet에 속할 Term을 결정하고 모델들간의 계층도를 작성하게 된다. 그러나 실제로 소프트웨어에 대한 Facet으로 제안된 예를 보면 Diaz에 의해 제안된 Function, Object, Media, Langauge, Environment등으로 이는 하나의 함수가 수행하는 기능성 또는 적용 대상을 표현하는 것이다. 즉, 모델에 대한 정형화된 Term 또는 Facet을 미리 정의해 둔다는 것은 매우 어려운 일이다. 또한 모델들간의 계층을 정의한 다는 것 역시 어려운 일이다. 즉 모델들간에는 개념적 또는 구현적 계층도를 정의하기 어렵다. 이런 관점에서 본다면 문제에 대한 개발자들의 의도를 표현하는 모델의 검색 또는 재사용 과정에서는 정형화된 틀을 정의하고 있는 Facet 기법과 계층적 분류 기법은 적용하기 어렵다. 이에 비해 매칭 기법이란 라이브러리에 잘 정립된 모델 또는 이미 검증된 모델을 저장후 사용자가 새로운 도메인에서 모델을 작성할 때 이와 유사한 모델을 매칭에 의해 검색하므로써 라이브러리에 저장된 모델을 기반 할 수 있도록 할 수 있다.

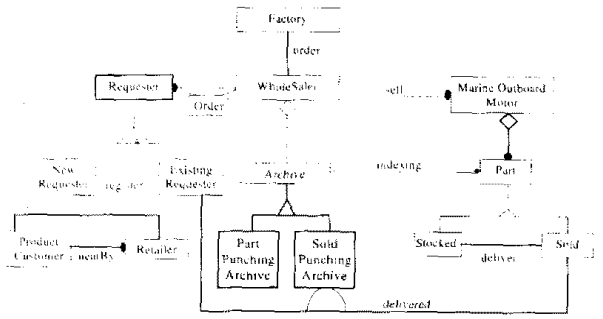
4.3.3 적용 결과

모델의 재사용 사례를 보이므로써 모델 재사용의 ... matching의 ... 설명하고자 한다. 이를 위해 서로 다른 두 어플리케이션을 사례로 선정하였다. 하나는 도서관의 논문 복사 서비스에 관한 문제이고, 다른 하나는 부품 판매상의 주문/배달 관리 시스템에 관한 문제이다. 본 논문의 실험에서는 부품 판매상의 주문/배달 관리 모델을 (그림 12-a)와 같이 라이브러리에 저장해 두었다. 이 두 문제는 서로 다른 어플리케이션이고, 기능적으로도 유사하지 않다. 도서관이 가지고 있는 서비스나 정보의 속성과 물류 창고가 가지고 있는 서비스와 정보의 속성들 사이에는 유사성이 없기 때문이다. 그러나 도서관이 도서를 가지고 있듯이 물류 창고도 물품을 가지고 있으며, 도서관에서 책을 빌려갈 고객이 있듯이 물류 창고에서 물건을 받아갈 고객이 있는 등 구조적으로는 대응성이 존재한다. 이러한 관점에서 보면 도서관과 물류 창고는 유사하다고 할 수 있다. 이를 위해 설계 라이브러리에 무엇인가를 대여하는 도메인에 대한 사전 정보를 저장해 두었다. 도서관 문제에 대한 개략적 모델을 (그림 12-b)와 같이 작성하였을 때 우리는 라이브러리에 저장된 (그림 12-a)의 모델을 검색할 수 있었고, 이를 바탕으로 도서관 문제에 대한 모델을 (그림 13)과 같이 재선하였다. 즉, 새로운 문제인 판매 어플리케이션에 관한 모델을 작성할 때 Requester의 개념을 세분화하거나 기관과의 연결 속성으로 Order를 식별하거나 장부의 개념을 식별하거나 대상과 요구사항의 서비스 관계를 식별하는 등의 모델링 정보를 재사용한 것이다. 그러나 판매 어플리케이션의 분석 기술시에는 창고에 존재하는 것과 이미 팔린 물건을 구분하는 개념이 기술되어 있어 판매 어플리케이션에 대한 모델 작성시 컴포넌트를 상세화 하는 서브 타입을 정의하였다. 이 개념을 바탕으로 (그림 13)과 같이 도서관 어플리케이션에 대한 모델을 수정할 수 있다. 즉, 도서관의 도서 가운데 이미 대출된 도서는 해당하는 논문을 복사할 수 없음을 보다 명시적으로 모델로 구축할 수 있게 된다. 이처럼 이미 구축된 모델이 기능적으로 유사성을 갖는 것은 아니지만 문제의 구조나 문제에 내포된 객체들의 행위적 관점에서 유사성이 있다면, 기존의 경험과 모델을 재사용 하는 것이 바람직하다. 예에서 보듯이 기존의 코드에 존재하는 개념이나 코드 재사용에 사용하던 분류 기법으로 라이브러리를 표현해서는 두 모델간에 개념적 재사용이 가능하다고 판단할 수 없다. 그러므로 객체 모델을 재사용 하

기 위해서는 analogy 개념에 기반한 매칭 기법을 통해 모델간의 개념적 구조와 행위의 유사성을 판단하여 재사용할 수 있도록 지원해야 한다.

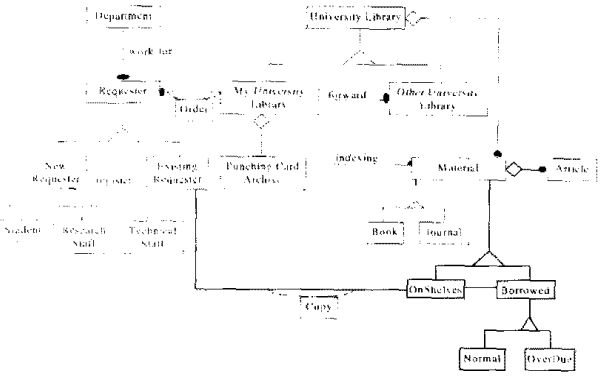


(a) Object Model for Library Application



(b) Object Model for Wholesale Application

(그림 12) 사례에 사용된 예제
(Fig. 12) Object Model for Case Study



(그림 13) 개선된 객체 모델
(Fig. 13) Revised Object model

5. 결 론

모델의 재사용은 여러 가지 장점을 갖는다. 모델은 문제에 대한 추상적인 개념을 보여 주는 것으로 해당 도메인에 대한 지식과 문제 해결 관점을 보여주므로써 새로 개발하고자 하는 문제에 대한 경험이 부족한 개발자들에게 해당 영역의 문제를 이해할 수 있도록 지원할 수 있다. 또한 모델링 과정에서 기존의 모델을 참조하므로써 이전의 모델링 경험을 재사용 하게 되고 따라서 사전에 모델이 잘못 구축되지 않도록 도와줄 수도 있다.

본 연구에서는 새로운 문제에 대한 객체 모델을 작성할 때 객체를 추출하는 과정과 객체들간의 관련성을 정립하는 과정에서 이전의 경험을 재사용할 수 있는 환경을 제공하기 위해 질의와 컴포넌트에 대한 analogy를 판단하여 라이브러리의 모델과 패턴을 재사용할 수 있는 방법을 제안하였다.

모델을 재사용 하기 위해 객체 모델의 최소 단위인 하나의 객체와 객체들간의 관련성을 포함하고 있는 패턴으로 재사용의 대상을 구분하였고 각각을 저장하는데 필요한 표현 기법을 정립하였다. 모델의 재사용을 위한 기법은 코드의 재사용 기법과는 달라서 구성 요소들간의 완전한 일치를 통해서만 유사도를 판단할 수는 없으므로, analogical matching 기법을 적용했다. 이를 위해 필요한 정보 구조와 매칭 판단에 필요한 함수를 재사용 대상별로 정의하였다. 또한 모델 작성 과정 중 사용자에게 부합되는 모델을 제공할 수 있는 환경을 조성하기 위해 analogical agent를 설계하였다. 개발자들은 당면한 문제에서 처음으로 식별한 객체를 중심으로 부분적 모델을 작성하고 이를 바탕으로 라이브러리에서 유사 객체를 검색할 수 있으며, 이로부터 객체를 식별하고 객체의 특성을 정의하는데 필요한 정보를 제공받을 수 있다. 이러한 과정을 반복하여 객체들을 식별하고 식별한 객체들간의 관련성을 정의하여 라이브러리에서 패턴 단위의 검색을 할 수 있다. 이렇게 검색된 유사한 패턴을 통해 유사한 문제들간의 모델 재사용을 지원할 수 있게 된다.

참 고 문 헌

[1] R. Breu, Algebraic Specification Techniques in Object Oriented Programming Environment, Spring-Verlag, 1991.

[2] Peter Deutsch, "Design reuse and framework in the Smalltalk-80 system," In Ted J. Biggerstaff and Alan J. Perlis, Editors, Software Reusability (II), 57-72, ACM Press, 1989.

[3] Ruben Prieto-Diaz, A Software Classification

Schema, Ph.D. thesis, University of Irvine, USA.
[4] Brian Falkenhainer, et. al., "The Structure Mapping Engine : Algorithm and Examples," Artificial Intelligence, 41, 1-63, 1989-90.
[5] Martin Fowler, Analysis Patterns, Reusable Object Models, Addison Wesley, 1997.
[6] Erich Gamma, et. al., Design Patterns : Elements of Reusable Object Oriented Software, Addison-Wesley, 1995.
[7] Gedre Gentner, "The mechanisms of analogical learning," In Bruce G. Buchanan and David C. Wilkins, editors, Readings in Knowledge Acquisition and Learning, Morgan Kaufmann Publishers, pp.673-694, 1993.
[8] Seymour Lipschutz, Discrete Mathematics, Mc Graw-Hill.
[9] Neil Arthur McDougall Maiden, Analogical Specification Reuse During Requirements Analysis, Ph.D. thesis, City University, London, July 1992.
[10] Bertrand Meyer, "Reusability : The case for object oriented design," In Ted J. Biggerstaff and Alan J. Perlis, editors, Software Reusability (II), 1-34, ACM Press, 1989.
[11] Tim O'Shea, "The learnability of object oriented programming systems," OOPSLA'86 proceedings, pp.502-504, Sep. 1986.
[12] Bruce W. Porter, et. al., "Concept Learning and Heuristic Classification in Weak-Theory Domains," Artificial Intelligence, 45, pp.229-263, 1990.
[13] James Rumbauch, et. al., Object Oriented Modeling and Design, Prentice-Hall, 1991.
[14] G. Salton and M. J. Mcgill, Introduction to Modern Information Retrieval, McGraw-Hill, 1983.
[15] R. Alan Whitehurst, Systemic Software Reuse Through Analogical Reasoning, Ph.D. thesis, University of Illinois at Urbana Champaign, 1995.
[16] George Wilkie, Object Oriented Software Engineering : The Professional Developer's Guide, Eddison-Wesley, 1993.
[7] 김영김, 소프트웨어 재사용을 위한 객체지향 클래스 라이브러리에서의 정보 검색, 박사 학위논문, 중대학교 컴퓨터공학과, 1992.

정 란

e-mail : jungran@samchook.ac.kr

1976년 중앙대학교 전자계산학과 졸업(공학사)

1983년 중앙대학교 대학원 전자계산학과 졸업(이학석사)

1998년 대구효성가톨릭대학교 대학원 전산통계학과 졸업(이학박사)

1983년~현재 삼척대학교 컴퓨터공학과 교수

관심분야 : 소프트웨어 공학, 소프트웨어 재사용, 지식공학

김 정 아

e-mail : clara@mail.kwandong.ac.kr

1988년 중앙대학교 전산과 졸업(이학사)

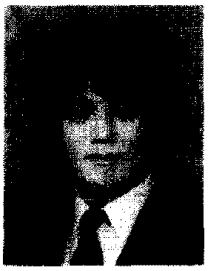
1990년 중앙대학교 대학원 전자계산학과 졸업(공학석사)

1994년 중앙대학교 대학원 전자계산학과 졸업(공학박사)

1994년~1996년 2월 중앙대학교 Post-Doc.

1996년 2월~현재 관동대학교 컴퓨터교육과 조교수

관심분야 : 소프트웨어 공학 재사용 이론, 형식 명세 기법, 객체지향 개발 방법론



김 행 곤

e-mail : hangkon@cuth.categu.ac.kr

1985년 중앙대학교 전자계산학과 (학사)

1987년 중앙대학교 대학원 전자계산학과(이학석사)

1991년 중앙대학교 대학원 전자계산학과(공학박사)

1978년~1979년 미 항공우주국 객원 연구원

1987년~1990년 한국전기통신공사 전임연구원

1988년~1989년 AT&T 객원 연구원

1990년~현재 대구효성가톨릭대학교 컴퓨터공학과 부교수

관심분야 : 객체지향 시스템 설계, 사용자 인터페이스, 소프트웨어 재공학, 유지보수 자동화 툴, CASE