工學碩士 學位論文

UML 모델의 관리를 위한 관계형 데이터베이스 설계 및 구현

Design and Implementation of a Relational Database for Management of UML Models

指導敎授 朴 烋 讚

2001年 2月

韓國海洋大學校 大學院

컴 퓨 터 工 學 科

李 聖 大

本 論文을 李聖大의 工學碩士 學位論文으로 認准함

委員長　工學博士　金　載　熏　印

委　員　工學博士　孫　周　永　印

委　員　工學博士　朴　烋　讚　印

2001年　1月

韓國海洋大學校 大學院

컴퓨터工學科　李　聖　大

목 차

Design and Implementation of a Relational Database for Management of UML Models

Seong Dae Lee

Department of Computer Engineering, Korea Maritime University, Pusan, Korea

Abstract

The UML (Unified Modeling Language) is a modeling language for specifying, visualizing, constructing, and documenting systems. It also supports systematically the design and development of systems. There may be a large number of models to be managed in a real modeling environment. A methodology to save and retrieve the models effectively, therefore, needs to be developed.

This thesis focuses on the class diagram which is the core part of UML. It proposes a database supported methodology for the management of class diagrams. In the proposed methodology, class diagrams are saved in and retrieved from a relational database. To save a class diagram in the database, the constitution of the class diagram is translated in terms of relational tables. To retrieve a class diagram from the database, the user-specified query is translated into the SQL (Structured Query Language), and then the constituent of the class diagram is searched from the tables in the

database.

The proposed methodology can exploit the function of the relational database such as managing a large number of models, sharing the models among users, and fast queries. It, therefore, provides a powerful framework for an effective management of UML models and a fast development of systems.

제 1 장 서론

최근 소프트웨어가 고부가가치 상품으로 인식되고 있고, 업무 처리 과정에서 소프트웨어가 처리할 업무의 범위와 규모가 커짐에 따라 시스템의 복잡성을 체계적으로 관리할 필요성이 증가되고 있다. 따라서 소프트웨어 생산을 자동화하고 개발에 필요한 시간과 비용을 절감하여 소프트웨어의 질을 향상시킬 수 있는 기술들이 모색되고 있다. 현재 이러한 기술로 부상하고 있는 것이 컴포넌트(component) 기술, 시각적 프로그래밍(visual programming) 그리고 패턴(pattern)과 프레임워크(framework) 등이 있다. 이 기술들은 물리적인 시스템의 분산(distribution), 동시성(concurrency), 반복성(replication), 보안(security), 결점보완 그리고 시스템의 부하에 대한 균등화(load balance)와 같은 반복해서 발생하는 구조적 문제를 해결할 수 있는 대안으로 제안되고 있다. 이 기술과 함께 소프트웨어 개발 분야에서는 개발 과정에서의 불필요한 수정과 보완을 미연에 방지하기 위해 개발할 시스템의 구성 요소를 체계적으로 분석하고 설계할 필요성이 있다. 이러한 필요성에 따라 개발된 것이 UML(Unified Modeling Language)이다.

UML은 분산 객체 어플리케이션 개발을 위한 공통 프레임워크 지침(framework guideline)과 상세한 객체 관리규약의 정립을 목표로 1989년 설립된 OMG(Object Management Group)에서 시스템 분석 및 설계 방법론의 표준 지정을 목표로 제안하였다. UML은 Rumbaugh의 OMT 방법론[1], Booch의 Booch 방법론[2,3], 그리고 Jacobson의 OOSE 방법론[4] 등 다수의 객체지향 방법론들을 통합하여 만든 통합 모델링 언어이다. 1997년 11월 UML 1.1을 시작으로 1999년 9월 UML 2.0 RFI[5]에 이르게 되었으며, 이미 많은 시스템 개발 도구들이 UML을 지원하고 있다.

현재 UML을 지원하는 대표적인 시스템으로 Rational사의 “Rose”[6]와 국내에서는 Plastic사의 “Plastic”[7]이 있다. 이 두 시스템은 공통적으로 파일 저장 시스템을 사용한다. 따라서 모델링 정보를 실시간으로 공유하는 것이 불가능하고, 설계에 다수의 사용자가 참여할 수 없기 때문에 효율적인 설계가 힘들다. 또한 이전에 모델링한 모델 정보의 검색이 어려워 유사한 모델을 다시 설계하는 경우가 발생하며, 운용 필요성에 따라서는 모델들을 통합하여 관리하는 것이 필요하지만 현재의 UML을 지원하는 도구에서는 어려움이 있다. 이러한 문제점을 해결하기 위해서는 UML 다이어그램 정보를 데이터베이스에 저장할 필요성이 있다.

UML 다이어그램을 관계형 데이터베이스에 저장하기 위해서는 각 다이어그램의 특성을 파악해야 한다. 본 논문에서는 UML의 구성요소 중 핵심인 클래스 다이어그램을 관계형 데이터베이스에 저장하고 검색하는 방법을 제안한다. 이를 위해서 클래스 다이어그램의 구성요소인 클래스(class), 속성(attribute), 연산(operation), 관계(relationship)를 각각 집합의 형태로 변환한다. 다시, 이러한 집합들을 관계형 데이터베이스 구조(relational schema)로 변환하고, 이 스키마들을 관계형 데이터베이스의 테이블들로 변환한다. 각각의 테이블들이 지니고 있는 키 정보(key information) 즉, 관계형 테이블의 기본키(primary key)와 참조키(foreign key)를 이용하여 테이블들이 서로 상호 연관성을 지니게 한다[8,9]. 이 상호 연관성에 의해서 사용자는 원하는 클래스 다이어그램의 정보를 검색할 수 있게 된다.

본 논문의 2장에서는 UML의 구성요소와 클래스 다이어그램 그리고 지금까지 진행된 관련 연구 및 관련 시스템에 대해서 살펴보고, 3장에서는 제안하는 UML 클래스 다이어그램의 관계형 데이터베이스로의 변환 및 저장 방법과 관계형 데이터베이스로부터 UML 클래스 다이어그램을 검색하는 방법에 대해서

설명한다. 4장에서 제안한 방법을 적용하여 구현한 UML 클래스 다이어그램 도구의 구성 및 소프트웨어 설계에 적용한 실행 결과를 설명하고, 기존 시스템과 비교한다. 5장에서는 결론과 함께 향후 연구과제에 대해서 논의한다.

제 2 장 UML 및 UML 개발 도구

UML은 통합된 시스템 개발방법론으로서 시스템을 가시화(visualizing), 명세화(specifying), 구조화(constructing) 그리고 문서화(documenting)하는 모델링 언어이다. 특히 기존에 존재하던 여러 개발 방법론들의 장점들을 모두 수용했으며, 확장이 용이하고, 다양한 표기법을 가지고 있어 실시간 시스템뿐만 아니라 여러 종류의 시스템 개발에 사용되어 질 수 있다.

본 장에서는 UML의 구성요소에 대하여 먼저 살펴보고, 다음으로 본 논문에서 다루게 될 클래스 다이어그램(Class Diagram)의 구성에 대해 설명한 후, 현재 클래스 다이어그램을 지원하는 개발 도구에 대해서도 살펴보도록 한다.

2.1 UML의 구성 요소

UML은 그림 2.1과 같이 크게 사물(Things), 관계(Relationships), 다이어그램(Diagrams)의 3가지 구성요소로 구분할 수 있다. 첫 번째 구성요소는 사물로서 구조 사물(Structural Things), 행동 사물(Behavioral Things), 그룹 사물(Grouping Things), 그리고 주해 사물(Annotation Things)로 나눌 수 있다. 구조 사물은 UML 모델의 실체를 나타내는 사물이고, 행동 사물은 동적인 부분을 나타낸다. 그룹 사물은 모델의 조직적인 부분을 나타낸다. 주해 사물은 모델에 대한 추가적인 설명을 표현하는 것이다. 두 번째 구성요소는 관계로서 의존(Dependency), 연관(Association), 일반화(Generalization), 실제화(Realization), 연관 관계의 확장 형태인 집합 연관(Aggregation) 관계로 나눌 수 있다. 마지막 구성요소로는 설계를 시각화하는 다이어그램이다. 다이어그램을 세분화하면 정적 구조(Static Structure), 사용 사례(Use Case), 순차(Sequence), 협력

(Collaboration), **상태도표**(Statechart), **활동**(Activity) **그리고 구현**(Implementation) **다이어그램이 있다. 다시, 정적 구조**(Static Structure) **다이어그램은 클래스**(Class), **객체**(Object) **다이어그램으로, 구현**(Implementation) **다이어그램은 컴포넌트**(Component), **배치**(Deployment) **다이어그램으로 세분할 수 있다**[10,11,12,13].

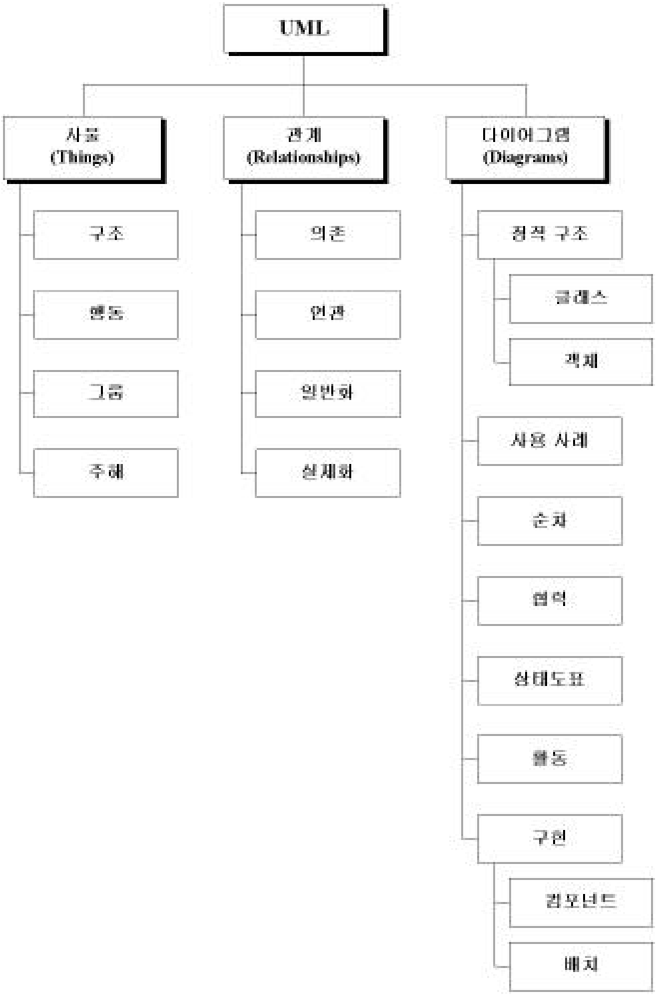

그림 2.1 UML **구성 요소**
Fig. 2.1 Building Blocks of UML

UML에서는 실제 시스템 구현을 위해서 몇 가지의 추가적인 방법론(mechanism)을 제공한다. 즉, 단순히 클래스의 상속 관계를 표현하는 것이 아니라 클래스 내부의 연산이 어떻게 실행되는가를 표현하는 것으로, 분석이나 설계의 내용을 사용자가 쉽게 이해하도록 하는 것이다. 종류로는 제한 조건(constraint)과 주석, 구성요소 특성(element property), 스테레오타입(stereotype)이 있다. 그 첫 번째로 제한 조건은 항상 참이 되어야하는 조건이나 상태를 기술하는 모델 사이의 의미적인 관계를 표현할 때 사용한다. '{'과 '}' 안에 문자열(text string)을 입력하여 사용한다. 예를 들어 '{ordered}'는 클래스 안의 요소들이 순서를 지켜야 하는 제한 조건이 된다. 주석은 모델 요소 사이에 직접 첨부되는 문자열로 이해를 돕는 단순한 설명이며 의미상 아무런 영향을 미치지 않는다. 두 번째, 구성요소 특성은 첨가되는 값(tagged value)이나 속성 등을 나타내는 것으로 클래스 이름 우측 하단에 '{ }' 표시를 한다. 세 번째, 스테레오타입은 기존의 속성이나 관계와 같은 모델링 요소와 같은 형태를 가지나 다른 취지로 사용할 하위 클래스로서 기본 클래스에 부가적인 제약 사항을 추가한 것이다. '≪'와 '≫' 사이에 문자열을 쓴다[5,6,10].

2.2 UML 클래스 다이어그램(Class Diagram)

UML의 많은 다이어그램들 중에서 가장 중심이 되는 것은 정적 구조 다이어그램으로 시스템을 논리적인 관점에서 분석한다. 여기에는 클래스 다이어그램과 객체 다이어그램이 있으며, 클래스 다이어그램은 클래스, 인터페이스(interface)와 같은 모델 요소와 모델 요소 간의 관계들로 구성되어 있으며, 정적인 구조 모델을 표현하는 다이어그램이다. 객체 다이어그램은 클래스 다이어그램의 인스턴스(instance)이다[5,6,10].

2.2.1 클래스(Class)와 인터페이스(Interface)

클래스 다이어그램은 클래스, 인터페이스 그리고 관계로 구성된다. 클래스는 비슷한 속성과 공통의 행위를 갖는 객체들의 그룹을 기술하는 단위로서 사물을 추상화하고 단순화하는 것이다[5,6,10]. UML에서는 그림 2.2와 같이 클래스를 표현하며, 계층적인 구조를 가지고 있다. 클래스는 이름(name)과 속성 목록(attribute list) 및 연산 목록(operation list)으로 구성된다. 각각의 속성이나 연산은 가시성(visibility) 표시를 가지는 데 '+'는 public, '-'는 private, '#'은 protected 속성을 나타낸다.

인터페이스는 연산(operation)들의 모임으로 하나의 클래스나 컴포넌트들이 제공하며, 서비스를 명세화하기 위해 사용한다. 인터페이스는 두 가지의 형태로 표현할 수 있다. 그 첫 번째는 그림 2.3의 (a)와 같이 원을 사용하여 단지 인터페이스임을 명시하는 방법이 있고, 두 번째는 그림 2.3의 (b)와 같이 클래스의 스테레오타입(stereotype)에서 인터페이스를 명시하는 방법이다.

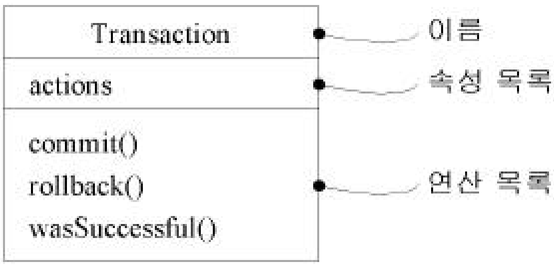

그림 2.2 UML 클래스의 표현
Fig. 2.2 Representation of UML Class

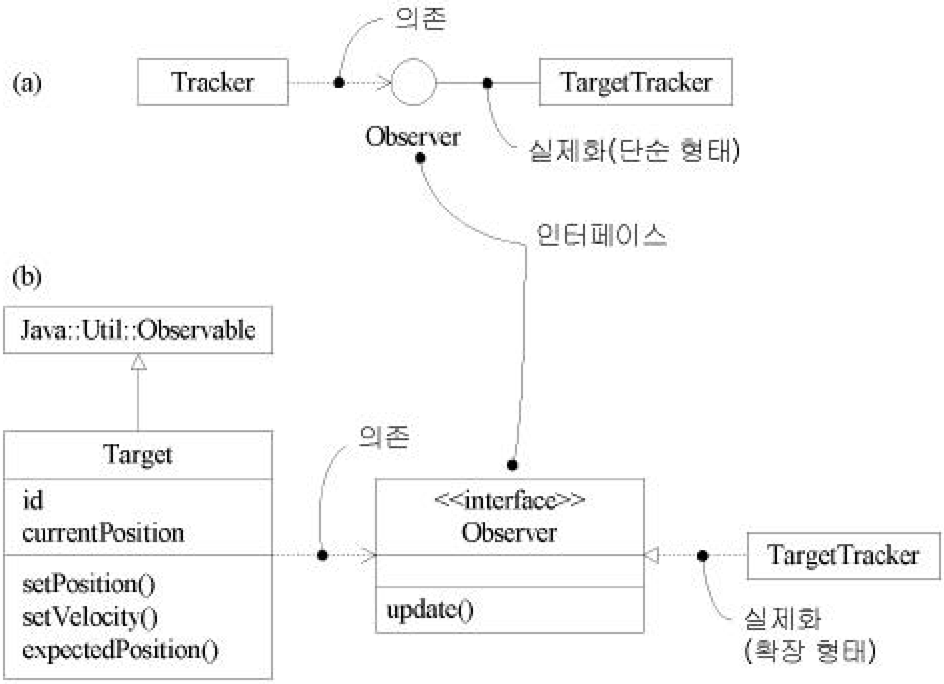

그림 2.3 UML **인터페이스의 표현**
Fig. 2.3 Representation of UML Interfaces

2.2.2 관계 (Relationship)

관계는 클래스와 클래스, 클래스와 인터페이스의 상호 연관성을 나타낸다. 관계에는 의존(dependency), **일반화**(generalization), **연관**(association), **집합 연관**(aggregation) **그리고 실제화**(realization) **관계가 있다. 의존 관계는 사용 관계로서 한 사물의 명세서가 바뀌면 이를 사용하는 다른 사물에게 영향을 끼치는 것을 의미한다. 즉, 사용 중인 클래스가 바뀌면 상대 클래스의 연산**(operation)**이 함께 영향을 받는다. 일반화 관계는** "is-a" **또는** "kind-of"**의 관계로서 상위**

클래스의 속성(attribute)과 연산(operation)이 하위 클래스로 상속되는 것을 의미한다. 연관 관계는 “has-a” 관계를 나타내며 구조적인 관계로서 특정 사물 객체가 다른 사물 객체와 관계가 있음을 표현하고, 역할(role)과 다중성(multiplicity)을 표시할 수 있다. 집합 연관 관계는 연관 관계의 확장형태로서 “part-of”의 관계이다. 실제화 관계는 인터페이스나 컴포넌트(component)와의 상호 연관성을 표시하는 관계이다.

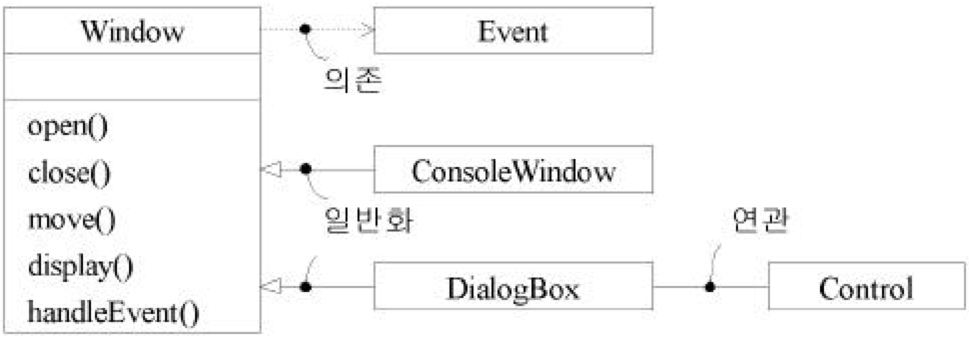

그림 2.4 클래스 다이어그램의 관계
Fig. 2.4 Relationship of Class Diagram

그림 2.4에서 클래스 “Window”는 클래스 “Event”와 의존 관계를 지니고 있으며, 클래스 “ConsoleWindow”와 “DialogBox”는 클래스 “Window”로부터 일반화, 즉 상속되었다. “Control” 클래스는 “DialogBox” 클래스와 연관 관계를 지니고 있음을 나타낸다.

앞에서 살펴본 것처럼 UML의 클래스 다이어그램은 각각의 구성요소들이 계층적으로 구조화되어 있음을 알 수 있다. 따라서 계층적으로 구조화된 요소들을 관계형 데이터베이스에 저장하기 위해서는 특별한 규칙을 정의해야 한다.

2.3 UML 개발 도구

소프트웨어를 설계하고 모델링하는 연구는 OMG를 중심으로 활발하게 진행되고 있다. 특히 객체 지향 방법으로 설계하는 것은 현재까지 크게 3세대로 구분할 수 있다. 1세대는 1980년대 후반 처음 객체 지향 개념이 도입되면서부터 1990년대 초반까지로, 기존의 여러 방법들이 혼합된 형태를 이루고 있다. 2세대는 1990년대 중반에 나타난 것으로 실제 현장에서 적용되고 계속적으로 연구 발전되었으며, 대표적인 것이 Booch94[3], OMT[1] 등이다. Booch94와 OMT 방법은 독립적으로 발전되었고 전세계적으로 가장 많이 사용하는 객체 지향 방법이 되었다. Booch, Rambaugh, Jacobson이 각각 자신들이 개발한 방법론들을 네 가지의 목적을 정하고 통합을 시작하였다. 그 첫 번째는 객체 지향 개념을 이용하여 단지 소프트웨어가 아닌 시스템을 모델하고, 두 번째는 실질적이고 개념적으로 기술들이 완벽하게 결합되도록 설정하고, 세 번째는 복잡하고 심각한 시스템의 계층적 크기 조절 문제를 해결하고, 마지막으로 사람과 기계 모두에게 이용 가능한 방법 즉, 수작업과 자동화가 동시에 가능한 방법을 만들자는 것이다. 이러한 통합의 결과를 UML이라 부르며, UML은 객체 지향 방법론에 의한 개발 과정 중의 결과물을 기술하고 시각화하며 문서화하는 3세대의 방법이다[14].

UML은 배우기 쉽고 확장하기 쉬우며 다양한 표기법을 지니고 있어서 여러 방법론의 특징을 잘 표현할 수 있다. 분석 단계에 만들어진 모델은 구현까지 연속성을 지니기 때문에 구현이나 설계에서의 변경은 곧 분석의 변경 사항이 되어 반복적인 방법을 통해 시스템을 개발할 수 있다.

UML을 지원하는 대표적인 시스템은 UML을 제안한 Booch, Rambaugh, Jacobson이 중심이 되어 개발한 “Rose”와 국내의 “Plastic”이 있다.

Rose는 현재 “Rational Rose Enterprise Edition 2000”이라는 이름으로 상용화된 소프트웨어이다. Rose에서 모델 정보를 저장하는 방법은 파일 저장 시스템을 이용한다. UML의 모든 다이어그램을 지원하며, 다이어그램을 C++, Java 등과 같은 객체 지향 언어의 코드를 생성시키는 순공학(forward engineering)과 객체를 표현하고 있는 코드에서 다이어그램을 생성하는 역공학(reverse engineering) 기능을 지원한다[6].

Plastic은 국내에서는 UML을 지원하는 유일한 도구로서 모델 정보의 저장은 파일 시스템을 사용하며 지원하는 다이어그램의 수는 객체(Object) 다이어그램을 제외한 총 8가지이다. 자바(java) 코드에 대한 순공학과 역공학을 지원한다[7].

제 3 장 UML 클래스 다이어그램의 저장 및 검색

UML 클래스 다이어그램은 다수의 클래스들로 구성되며, 각 클래스 내부에는 다수의 속성(attribute)과 연산(operation)이 존재하는 계층적인 구조로 표현된다. 하지만 관계형 데이터베이스에서는 이러한 계층적인 구조를 직접적으로 허용하지 않으므로 클래스 다이어그램을 관계형 스키마로 변환하기 위해서 다수의 규칙들을 정의해야 한다. 또한, 이러한 스키마에 따라 저장된 데이터베이스에서 클래스 다이어그램을 검색하기 위해서는 역시 검색의 방법 및 순서를 설정해야 한다.

3.1 UML 클래스 다이어그램 변환

모델링하고자 하는 시스템에 대한 모델은 여러 개의 클래스 다이어그램으로 구성된다. 각 클래스 다이어그램은 클래스, 속성, 연산, 관계 등으로 구성되며 이들을 각각의 특징을 고려하여 관계형 스키마로 변환하기 위해서 다음과 같은 규칙들을 정의한다.

[규칙 1] 클래스 다이어그램 집합 $D = \{d_1, d_2, ..., d_k\}$으로 정의하고, D의 임의의 원소 d_x는 클래스 다이어그램의 식별자이다(단, $x = 1, 2, ..., k$).

본 논문에서는 식별자로서 다이어그램의 이름을 사용하고 실제 구현에서는 각 클래스 다이어그램의 ID를 생성하여 사용한다. 집합 D는 전체 클래스 다이어그램 목록을 담고 있는 집합이다. 구현부분에서는 이 목록과 함께 각 클래스 다이어그램에 대한 정보도 저장되도록 구현한다.

[규칙 2] 클래스 다이어그램 내의 클래스 집합 $C = \{c_1, c_2, ..., c_l\}$**으로 정의하고,** C**의 임의의 원소인 클래스** $c_x = (class\_name, d_i)$**으로 정의한다. 여기서,** d_i**는** c_x**가 소속된 클래스 다이어그램** D**의 원소이다(단,** $x = 1, 2, ..., l$).

규칙 2는 각 클래스 다이어그램 내부에 존재하는 클래스의 집합에 관한 것이다. 따라서 집합 C의 원소는 식별자로 사용할 클래스 이름과 이 클래스들이 어떤 다이어그램에 소속되어 있는가를 나타낸다.

[규칙 3] 클래스 내부의 속성 집합 $A = \{a_1, a_2, ..., a_m\}$**으로 정의하고,** A**의 임의의 원소인 속성** $a_x = (attribute\_name, c_i)$**으로 정의한다. 여기서,** c_i**는** a_x**가 소속된 클래스** C**의 원소이다(단,** $x = 1, 2, ..., m$).

[규칙 4] 클래스 내부의 연산 집합 $O = \{o_1, o_2, ..., o_n\}$**으로 정의하고,** O**의 임의의 원소인 연산** $o_x = (operation\_name, c_i)$**으로 정의한다(단,** $x = 1, 2, ..., n$).

규칙 3과 4에서 집합 A는 클래스 내부의 속성으로, 집합 O는 연산으로 구성된 집합으로 표현할 수 있다. 각각의 집합의 원소는 역시 식별자와 함께 소속된 클래스를 나타낸다.

[규칙 5] 클래스 다이어그램 내부의 인터페이스는 클래스의 스테레오타입을 나타내는 것으로 클래스의 집합 C**를 확장하여 표현한다. 즉,** C**의 임의의 원소인 클래스** c_x**를** $c_x = (class\_name, is\_interface, d_i)$**로 확장하여 정의한다. 추가된 항목** *is_interface***는 인터페이스 여부를** TRUE, FALSE**로 표시한다.**

인터페이스는 그림 2.3과 같이 두 가지 형태로 표현된다. 따라서 다른 스테레오타입과 구분하기 위하여 인터페이스임을 구분하는 필드가 필요하다.

실제 구현을 위해서는 위에서 정의한 규칙들을 확장해야 한다. 확장에 필요한 부분으로는 규칙 3에서 속성의 타입이나 규칙 4의 연산의 반환 타입(return type), 인자(parameter)에 대한 정의 등이 있을 수 있다. 확장에 필요한 부분들은 실제 구현에서 고려하였다.

각 클래스의 상속성과 연관성을 설정하는 관계의 집합은 다음과 같이 정의할 수 있다.

[규칙 6] 클래스 다이어그램 내부의 클래스간의 관계 집합 $R = \{r_1, r_2, \ldots, r_o\}$ **으로 정의하고,** R**의 임의의 원소인 관계** $r_x = (c_i, c_j, relationship, d_i)$**으로 정의한다**(단, $x = 1, 2, \ldots, o$).

c_i와 c_j는 각각 클래스를 의미하고, 두 클래스가 $relationship$의 관계가 있음을 나타낸다. 여기서, c_i는 상위 클래스(super class)이고 c_j는 하위 클래스(sub class)이다.

그림 3.1은 “Company”의 구성을 UML로 모델링한 예제 클래스 다이어그램이다. 그림 3.1에서 검정으로 채워진 다이아몬드 화살표는 연관 관계의 확장 형태인 집합 연관(aggregation) 관계를 의미한다. 클래스 Company는 클래스 Department, Office와는 집합 연관 관계를 지니고 있다. 이 관계에서 Company의 다중성(multiplicity)은 “1”이고, Department, Office의 다중성은 “1..*”이다.

즉, 하나의 Company는 하나 이상의 Department와 Office의 집합으로 구성된다는 것을 의미한다. 클래스 Department는 클래스 Person과 두 개의 연관 관계를 가진다. 첫 번째 연관 관계의 역할명(role name)은 “member”이고, 두 번째는 “manager”이다. 이 두 연관 관계에는 제약 조건 “subset”이 있다. 즉, “manager”는 “member”의 부분 집합이다라는 제약 조건을 가진다는 것을 의미한다.

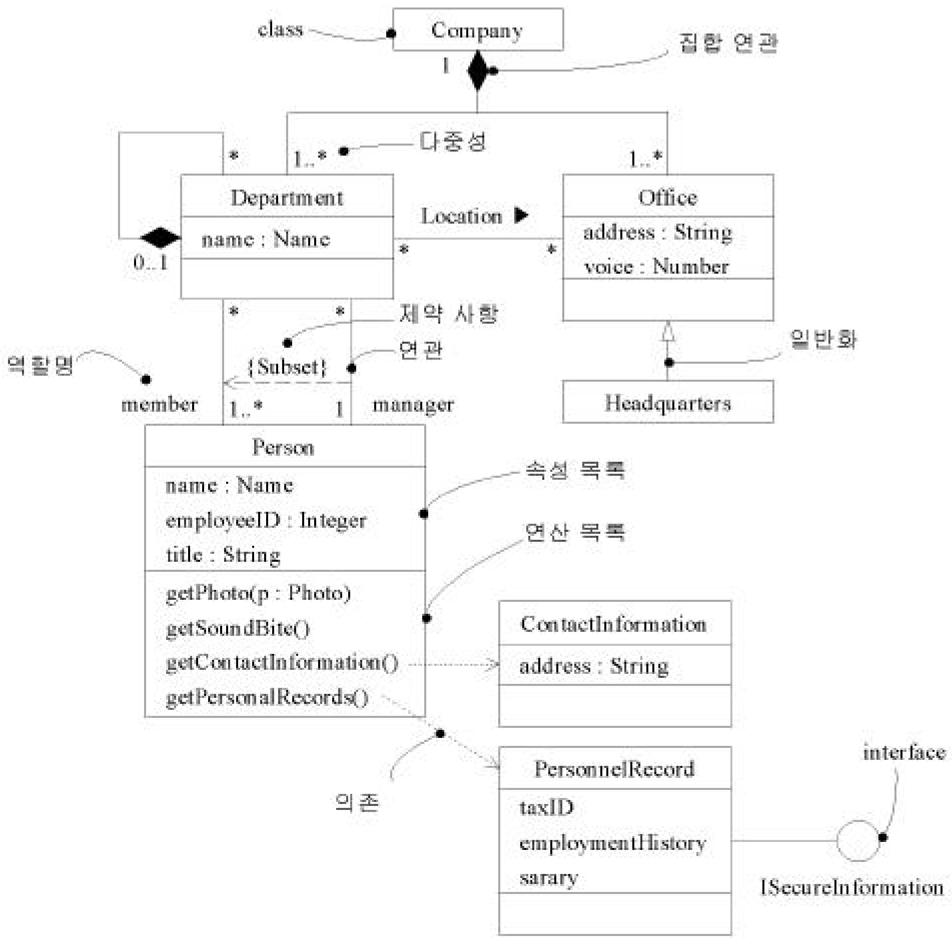

그림 3.1 예제 클래스 다이어그램 : *Company*
Fig. 3.1 An Example of a Class Diagram : *Company*

(예제 1) 그림 3.1**의 클래스 다이어그램에 본 논문에서 정의한 규칙을 적용하면 다음과 같이** 5**개의 집합들을 구할 수 있다**.

- **클래스 다이어그램 집합**

  D = {Company}

- **클래스 집합**

  C = {(Company, False, Company),
  (Department, False, Company),
  (Office, False, Company),
  (Person, False, Company),
  (ContactInformation, False, Company),
  (PersonnelRecord, False, Company),
  (Headquarters, False, Company),
  (ISecureInformation, True, Company)}

- **속성**(attribute) **집합**

  A = {(name, Department),
  (address, Office),
  (voice, Office),
  (name, Person),
  (employeeID, Person),
  (title, Person),
  (address, ContactInformation),
  (taxID, PersonnelRecord),
  (employmentHistory, PersonnelRecord),
  (salary, PersonnelRecord)}

、 **연산**(operation) **집합**

O = {(getPhoto, Person),
(getSoundBite, Person),
(getContactInformation, Person),
(getPersonnelRecord, Person)}

、 **관계 집합**

R = {(Company, Department, aggregation, Company),
(Company, Office, aggregation, Company),
(Department, Department, aggregation, Company),
(Department, Office, association, Company),
(Department, Person, association, Company),
(Department, Person, association, Company),
(Office, Headquarters, generalization, Company),
(Person, ContactInformation, dependency, Company),
(Person, PersonnelRecord, dependency, Company),
(PersonnelRecord, ISecureInformation, association, Company)}

3.2 관계형 데이터베이스 생성 및 저장

규칙 1 6**을 적용하여 생성된** 5**개의 집합을 관계형 모델의 관계형 스키마로 변환하고**, **관계형 테이블로 변환하기 위해서 다음의 규칙들을 추가로 정의한다**.

[규칙 7] 각 집합 D, C, A, O, R**은 관계형 모델**(relational model)**에서 각각 하나의 관계형 데이터베이스 구조**(relational schema)**로 변환한다**.

규칙 7을 적용하면 위에서 정의한 규칙 1 6은 다음과 같은 다섯 개의 관계형 스키마로 변환된다.

$$D\text{-}schema = (diagram\_name)$$

$$C\text{-}schema = (class\_name,\ is\_interface,\ diagram\_name)$$

$$A\text{-}schema = (attribute\_name,\ class\_name)$$

$$O\text{-}schema = (operation\_name,\ class\_name)$$

$$R\text{-}schema = (super\_class,\ sub\_class,\ relationship,\ diagram\_name)$$

[규칙 8] 각 집합의 원소는 규칙 7을 통하여 생성된 관계형 스키마의 튜플 (tuple)로 변환된다.

규칙 1 8을 적용하면 클래스 다이어그램의 정보를 가지는 5개의 테이블이 생성된다. 본 장에서는 각 집합 내 원소들을 구별하기 위하여 규칙들을 집합의 원소 수준까지 확장하였고, 관계형 테이블에서 튜플을 식별할 수 있는 최소 단위의 속성으로만 테이블을 구성하였다. 실제 시스템을 구현하기 위해 필요한 테이블의 속성들은 다음 장에서 추가하도록 한다.

다음의 그림 3.2는 위의 관계형 스키마를 사용하여 그림 3.1의 *Company* 예제를 관계형 테이블로 변환한 것이다.

D (Diagram) Table

diagram_name
Company

C (Class) Table

class_name	*is_interface*	*diagram_name*
Company	False	Company
Department	False	Company
Office	False	Company
Person	False	Company
ContactInformation	False	Company
PersonnelRecord	False	Company
Headquarters	False	Company
ISecureInformation	True	Company

O (Operation) Table

operation_name	*class_name*
getPhoto	Person
getSoundBite	Person
getContactInformation	Person
getPersonnelRecord	Person

A (Attribute) Table

attribute_name	*class_name*
name	Department
address	Office
voice	Office
name	Person
employeeID	Person
title	Person
address	ContactInformation
taxID	PersonnelRecord
employmentHistory	PersonnelRecord
salary	PersonnelRecord

그림 3.2 변환된 데이터베이스 테이블
Fig. 3.2 Converted Database Tables

R (Relationship) Table

super_class	*sub_class*	*relationship*	*diagram_name*
Company	Department	aggregation	Company
Company	Office	aggregation	Company
Department	Department	aggregation	Company
Department	Office	association	Company
Department	Person	association	Company
Department	Person	association	Company
Office	Headquarters	generalization	Company
Person	ContactInformation	dependency	Company
Person	PersonnelRecord	dependency	Company
PersonnelRecord	ISecureInformation	association	Company

그림 3.2 변환된 데이터베이스 테이블 - 계속

Fig. 3.2 Converted Database Tables - continued

3.3 UML 클래스 다이어그램 검색

변환 규칙을 적용하여 저장된 데이터베이스의 각 테이블로부터 클래스 다이어그램을 검색하기 위해서는 다이어그램 테이블에서 사용자가 요구하는 다이어그램의 식별자를 먼저 검색해야 한다. 임의의 클래스 다이어그램이 주어졌을 때, 관계형 데이터베이스에서 SQL을 사용하여 클래스 다이어그램을 검색하는 방법을 순차적으로 설명하면, 먼저 주어진 클래스 다이어그램의 식별자를 검색어로 사용하여 소속된 클래스와 인터페이스들을 검색한다. 다음은 각 클래스 내부의 속성들과 연산들을 검색한다. 마지막으로, 각 클래스 및 인터페이스 간의 관계들을 검색함으로써 원하는 클래스 다이어그램을 전체적으로 검색할 수 있다. 클래스 다이어그램 외의 정보 즉, 클래스 다이어그램에 대하여 사용자가 직접 입력한 정보나 클래스 이름, 속성 또는 연산의 이름으로 검색하는 경우는 먼저 소속된 클래스 다이어그램의 식별자를 추출한 후 위의 방법을 적용한다.

```
// $operation_name이라는 연산(operation)의 이름이 주어진 경우
// 결과 값은 클래스 다이어그램의 식별자 $diagram_name이다.

SELECT O.class_name INTO $class_name
FROM O
WHERE O.operation_name = $operation_name

// 클래스 선택
for each $class_name
    SELECT C.diagram_name INTO $diagram_name
    FROM C
    WHERE C.class_name = $class_name
end for
```

위와 같이 클래스 다이어그램의 식별자를 검색하는 과정은 몇 개의 SQL을 실행함으로서 가능하다. 클래스 다이어그램의 식별자가 검색된 이후는 다음과 같은 과정을 수행하면 된다.

```
// 주어진 클래스 다이어그램의 식별자는 $diagram_name이다.
SELECT C.class_name INTO $class_name
FROM C
WHERE C.diagram_name = $diagram_name

for each $class_name
    // 각 클래스 내부의 속성들을 검색한다.
    SELECT A.attribute_name
    FROM A
    WHERE A.class_name = $class_name

    // 각 클래스 내부의 연산들을 검색한다.
    SELECT O.operation_name
    FROM O
    WHERE O.class_name = $class_name
end for

// 다이어그램에 소속된 클래스간의 관계들을 검색한다.
SELECT R.relationship, R.super_class, R.sub_class
FROM R
WHERE R.diagram_name = $diagram_name
```

추가적으로 UML 클래스 다이어그램의 보안을 위하여 설계자의 정보를 테이블로 구성하여 관리한다. 실제 구현을 위한 테이블의 상세 설계는 4장에서 다시 언급하도록 한다.

제 4 장 구현 및 기존 시스템과의 비교

본 장에서는 UML 클래스 다이어그램을 관계형 데이터베이스로 저장하고 검색하는 알고리즘에 대해서 먼저 설명하고, 이 알고리즘을 이용하여 구현한 시스템의 전체적인 흐름도와 실제 구현에 필요한 관계형 데이터베이스 테이블들에 대해서 설명한다. 또한 본 논문에서 제안한 방법을 적용하여 구현한 시스템을 소개하고, 현재 UML을 지원하는 기존 시스템과 비교한다.

4.1 알고리즘 및 시스템 흐름도

아래의 알고리즘은 관계형 데이터베이스로부터 UML 클래스 다이어그램을 검색하는 과정과 UML 클래스 다이어그램을 관계형 데이터베이스로 저장하는 과정을 기술한 것이다. 검색에서의 입력은 클래스 다이어그램의 이름을 기준으로 하였고, 출력은 전체 클래스 다이어그램이 된다. 저장을 위한 입력은 클래스 다이어그램을 구성하는 전체 구성 요소가 되고, 결과로서 각각의 테이블에 데이터가 저장된다.

```
begin
{
  // 클래스 다이어그램의 이름은 사용자 입력
  $diagram_name = input_diagram_name();
  // 현재 수행하고자 하는 작업이 클래스 다이어그램 검색이면
  if (diagram_search == TRUE)
  {
    // 입력받은 다이어그램에 소속된 모든 클래스와 인터페이스 검색
    $class_name[] = select_all_class($diagram_name);
    for each $class_name     // 각각의 클래스에 대하여
```

```
    {
      select_all_attribute($class_name);     // 소속 속성 검색
      select_all_operation($class_name);     // 소속 연산(operation) 검색
    }
    select_all_relationship($diagram_name); // 소속 관계 검색
  }
  // 현재 수행하고자 하는 작업이 클래스 다이어그램 저장이면
  else
  {
    // $class_name이 NULL일 때까지
    while (not class_name_set_empty())
    {
      // 클래스 이름 및 관련된 정보 저장
      insert_class_name($class_name, $diagram_name);
      // 현 클래스 내부의 속성 목록(attribute list) 저장
      while (not attribute_list_empty())
        insert_attribute_name($attribute_name, $class_name);
      // 현 클래스 내부의 연산 목록(operation list) 저장
      while (not operation_list_empty())
        insert_operation_name($operation_name, $class_name);
      // 현재 다이어그램 내부에 있는 모든 관계 검색, 저장
      if (relationship_seek())
        insert_relationship($ret_class_name, $class_name, $diagram_name);
    }
  }
}
end
```

그림 4.1은 본 논문에서 제안한 알고리즘을 구현한 시스템의 전체 흐름도(flowchart)이다. UML의 클래스 다이어그램을 관계형 데이터베이스로 저장하기 위한 시스템의 흐름도와 관계형 데이터베이스로부터 클래스 다이어그램 정보를 검색하는 흐름도를 나타내고 있다. 클래스 다이어그램의 보안을 위하여 사용자 로그인 과정이 필요하다.

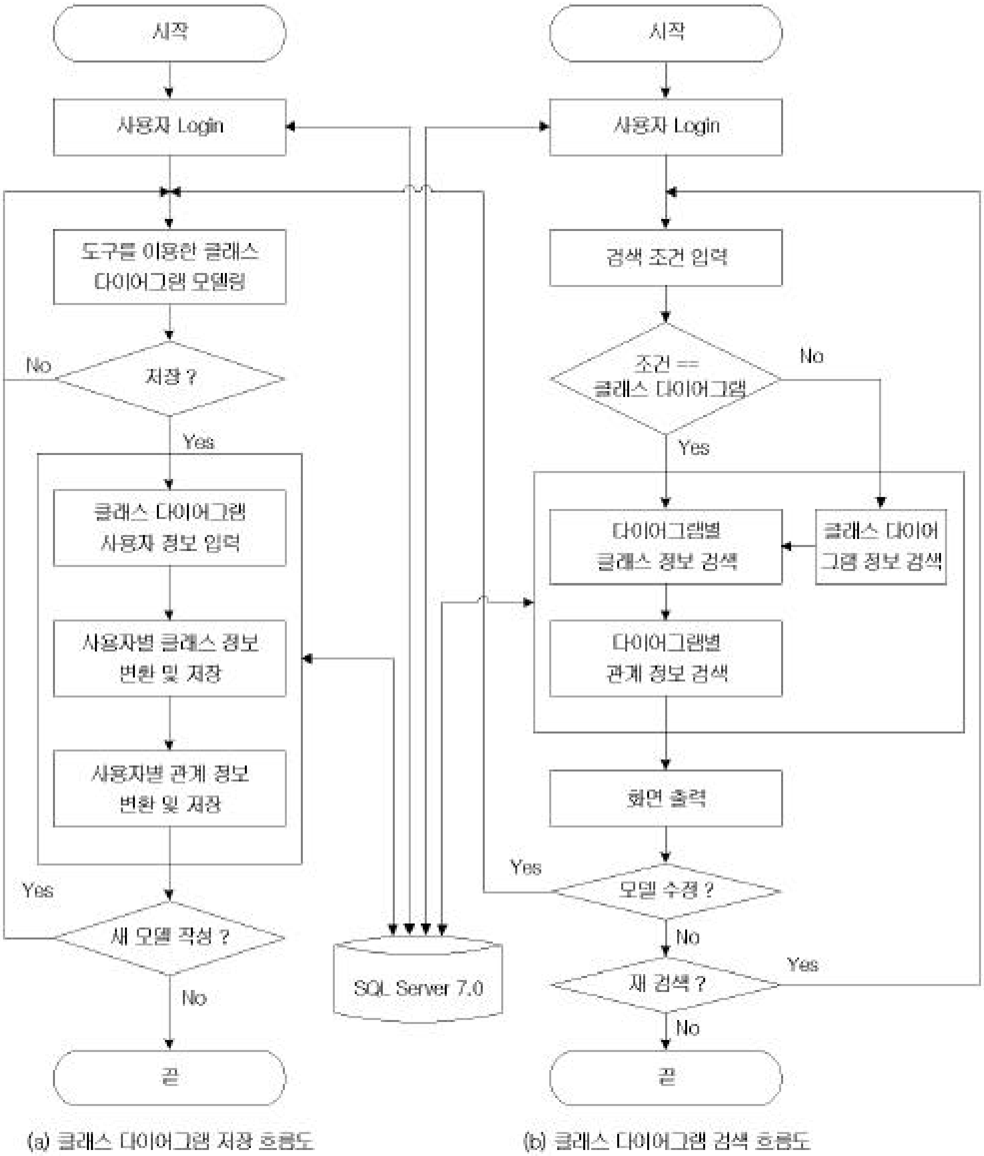

그림 4.1 클래스 다이어그램 저장 및 검색 흐름도

Fig. 4.1 Flowchart for Storing and Retrieving Class Diagram

4.2 데이터베이스 테이블 상세 설계

3장에서 제시한 관계형 테이블에 실제 구현을 위해 필요한 부분들을 추가하여 확장한 테이블을 생성한다. 표 4.1 4.9는 실제 구현에서 사용한 관계형 데이터베이스 테이블의 구조이다. 표의 기타 칼럼에서 "P-K"는 기본 키(primary key)를 "F-K"는 참조 키(foreign key)를 의미한다. 각 키 정보를 이용하여 테이블들의 관계를 도식화하면 그림 4.2와 같다. 그림 4.2는 Microsoft SQL Server 7.0에서 지원하는 E-R(Entity-Relationship) 다이어그램이다.

표 4.1 다이어그램 저장 테이블 구조 : Diagram
Table 4.1 Table Structure for Storing Diagrams : Diagram

이름	타입	길이	Null	기본값	초기값	증가값	기타
Diagram_ID	int	4	×	ID	1	1	P-K
Diagram_Name	varchar	50	×				이름
Diagram_Info	varchar	50	×				기타 정보
Diagram_Owner	char	10	×				F-K
Diagram_FDate	datetime	8	×	getdate()			현재 일자
Allow_Others	bit	1	×	0			공개 여부

표 4.1은 클래스 다이어그램 정보를 저장하는 테이블로서 클래스 다이어그램을 작성한 사용자(Diagram_Owner)와 정보의 공개 유무(Allow_Others)를 갖는 필드를 추가하였다. 사용자는 다이어그램을 검색할 경우 자신이 작성한 모델과 다른 사람이 작성한 공개 모델을 검색할 수 있다. 또한 사용자가 임의로 입력하는 정보(Diagram_Info)를 추가함으로써 이 후 검색의 효율성을 높일 수 있도록 하였다.

표 4.2 클래스 저장 테이블 구조 : Class
Table 4.2 Table Structure for Storing Classes : Class

이름	타입	길이	Null	기본값	초기값	증가값	기타
Class_ID	int	4	×	ID	1	1	P-K
Class_Name	varchar	50	×				이름
Class_Info	varchar	50	○				기타 정보
Diagram_ID	int	4	×				F-K
Is_Interface	bit	1	×	0			인터페이스
Stereotype	varchar	20	○				스테레오타입
Start_X	int	4	×				시작 X, Y 좌표
Start_Y	int	4	×				

표 4.2는 클래스의 정보를 저장하는 테이블로서 소속 다이어그램을 참조키(foreign key)로 지정하여 잘못된 데이터의 입력을 미연에 방지할 수 있도록 하였다. 클래스를 검색한 후, 화면 출력을 위하여 시작 좌표(StartX, StartY)를 추가로 저장하며 화면 출력에서 클래스의 크기는 텍스트의 크기에 비례하여 자동으로 증가하도록 하였다. 또한, 사용자가 입력하는 클래스 정보(Class_Info)를 저장함으로써 클래스 이름 외의 기타 정보를 사용하여 검색할 수 있도록 하였다.

표 4.3 속성 저장 테이블 구조 : Attribute
Table 4.3 Table Structure for Storing Attributes : Attribute

이름	타입	길이	Null	기본값	초기값	증가값	기타
Attr_ID	int	4	×	ID	1	1	P-K
Attr_Name	varchar	50	×				이름
Attr_Visibility	int	4	○				가시성 번호
Attr_Type	varchar	20	○				타입
Attr_Stereotype	varchar	20	○				스테레오타입
Class_ID	int	4	×				F-K
Attr_Info	varchar	50	○				기타 정보

표 4.4 연산 저장 테이블 구조 : Operation
Table 4.4 Table Structure for Storing Operations : Operation

이름	타입	길이	Null	기본값	초기값	증가값	기타
Oper_ID	int	4	×	ID	1	1	P-K
Oper_Name	varchar	50	×				이름
Oper_Type	varchar	20	○				반환 타입
Oper_Para	varchar	50	○				인자 목록
Oper_Visibility	int	4	○				가시성 번호
Oper_stereotype	varchar	20	○				스테레오타입
Class_ID	int	4	×				F-K
Oper_Info	varchar	50	○				기타 정보

표 4.3과 4.4는 클래스 내부의 속성과 연산(operation)을 저장하는 테이블로서, 표 4.2의 Class 테이블의 기본키인 Class_ID를 참조하여 생성된다.

표 4.5 관계 저장 테이블 구조 : Relationship
Table 4.5 Table Structure for Storing Relationships : Relationship

이름	타입	길이	Null	기본값	초기값	증가값	기타
Relationship_ID	int	4	×	ID	1	1	P-K
Super_Class	int	4	×				F-K
Sub_Class	int	4	×				F-K
Relationship	int	4	×				관계 번호
Relationship_Info	varchar	50	○				기타 정보
Super_multi	char	10	○				다중성 표시
Sub_multi	char	10	○				
Super_Rolename	varchar	50	○				역할명 표시
Sub_Rolename	varchar	50	○				

표 4.5는 클래스나 인터페이스와 같은 모델 요소 간의 관계를 저장하는 테이블이다. 관계와 연결된 클래스의 다중성이나 역할명은 연관(Association) 또는 집합 연관(Aggregation) 관계인 경우에 사용된다.

표 4.6 사용자 저장 테이블 구조 : Users
Table 4.6 Table Structure for Storing Users : Users

이름	타입	길이	Null	기본값	초기값	증가값	기타
ID	int	10	×				P-K
Password	varchar	10	×				암호
FName	char	10	×				이름
LName	char	10	×				성

표 4.6의 Users 테이블은 모델 정보의 보안을 위하여 사용되는 테이블로서, 등록된 사용자만이 클래스 다이어그램을 저장하고 검색할 수 있도록 한다.

표 4.7 제약사항 저장 테이블 구조 : Constraint
Table 4.7 Table Structure for Storing Constraints : Constraint

이름	타입	길이	Null	기본값	초기값	증가값	기타
Constraints_ID	int	4	×	ID	1	1	P-K
Relation_X	int	4	×				F-K
Relation_Y	int	4	×				F-K
Constraints_Name	varchar	20	×				이름
Constraints_Info	varchar	50	○				기타 정보

표 4.7은 제약사항을 저장하는 테이블로서 그림 3.1의 두 관계 "member"와 "manager"간의 "{subset}" 제약사항을 표시할 때 사용된다.

표 4.8 Note 저장 테이블 구조 : Note
Table 4.8 Table Structure for Storing Notes : Note

이름	타입	길이	Null	기본값	초기값	증가값	기타
Note_ID	int	4	×	ID	1	1	P-K
Note_Text	varchar	50	×				내용
Note_Flag	int	4	×				구분
Start_X	int	4	×				시작 X, Y 좌표
Start_Y	int	4	×				
Diagram_ID	int	4	×				F-K

표 4.8은 다이어그램에서 주석 부분을 저장하는 테이블로서 클래스 다이어그램에서 특별한 의미를 가지지는 않지만 사용자 편의를 위해 필요한 부분이다.

표 4.9 **링크 저장 테이블 구조** : Link

Table 4.9 Table Structure for Storing Links : Link

이름	타입	길이	Null	기본값	초기값	증가값	기타
Link_ID	int	4	×	ID	1	1	P-K
Class_ID	int	4	×				F-K
Note_ID	int	4	×				F-K
Diagram_ID	int	4	×				F-K

표 4.9**는 주석과 클래스를 연결하는 링크 정보를 저장하는 테이블이다**.

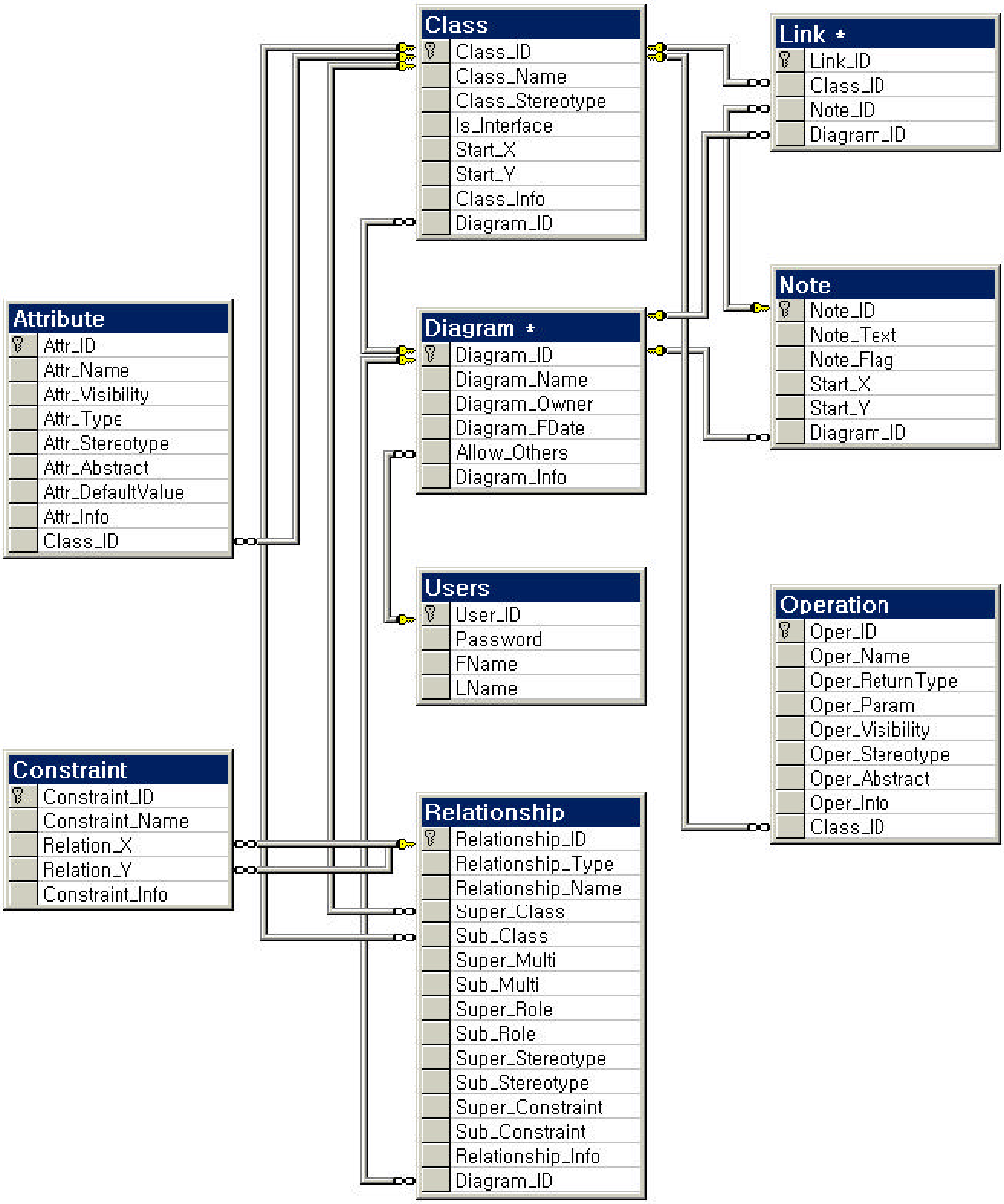

그림 4.2 클래스 다이어그램 관리를 위한 관계형 테이블의 E-R Diagram

Fig. 4.2 E-R Diagram of Relational Tables for Managing Class Diagram

4.3 시스템 구현

본 논문에서 제안한 방법을 구현하기 위하여 저장과 검색을 위한 관계형 데이터베이스 관리 시스템(Relational Database Management System)으로는 Microsoft SQL Server 7.0을 사용하였고, 프로그래밍 언어로는 Visual C++ 6.0을 사용하였다. ODBC(Open Database Connectivity)를 통해서 데이터베이스와 연결하고, 트랜잭션(transaction)을 처리하기 때문에 관계형 데이트베이스가 바뀌는 경우는 ODBC의 설정을 변경하면 구현한 시스템을 쉽게 사용할 수 있다. 그림 3.1의 예제 “Company” 클래스 다이어그램을 실제 구현한 시스템에서 모델링한 화면은 그림 4.3이다.

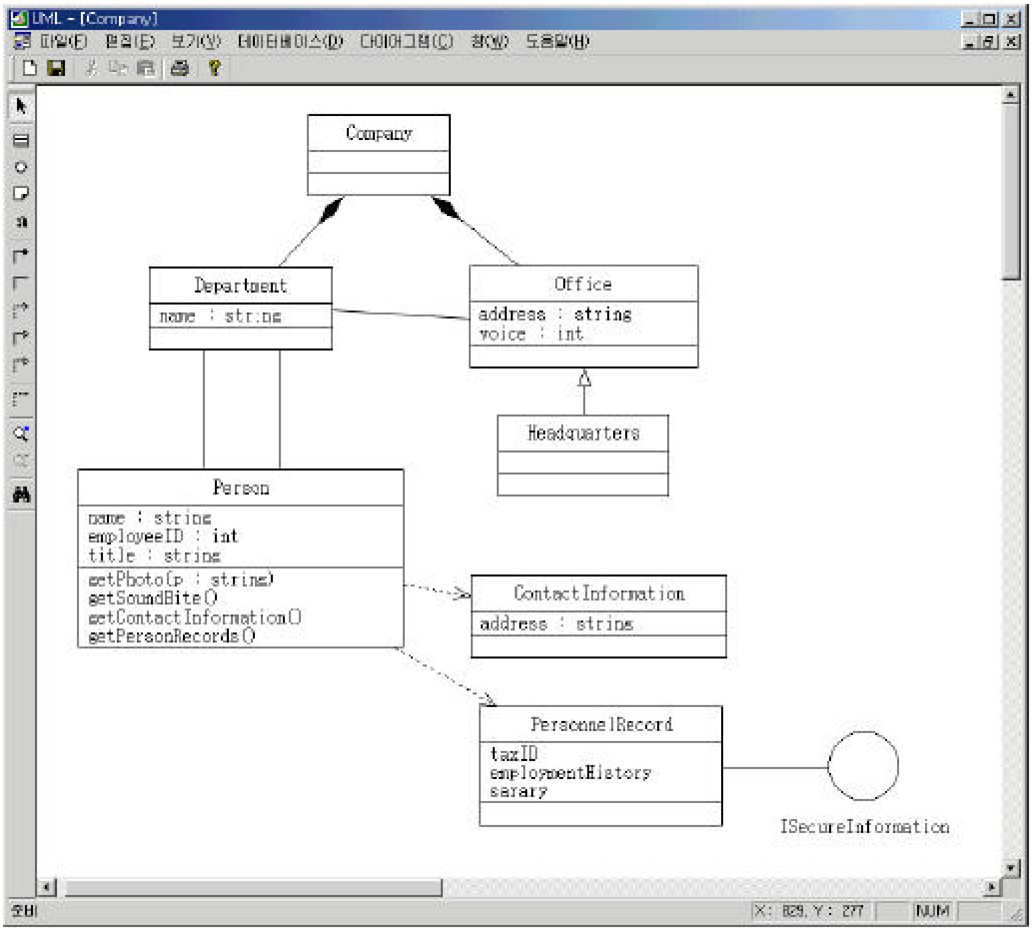

그림 4.3 구현 프로그램의 예제

Fig. 4.3 An Example of Implemented Program

4.4 기존 시스템과의 비교

제안한 시스템은 UML을 사용하여 설계된 모델링 정보를 다수의 사용자들이 실시간으로 공유하여 설계에 공동으로 참여할 수 있도록 하기 위해 기존의 파일 저장 시스템이 아닌 관계형 데이터베이스를 사용하도록 하였다. 기존에 파일 저장 시스템을 사용하고 있는 Rose나 Plastic은 UML에서 지원하는 모든 다이어그램을 지원하고 있으며, 순공학과 역공학을 지원하지만 실시간에 모델링 정보를 공유할 수 없다는 단점이 있다. 표 4.10은 Rose, Plastic 그리고 제안한 시스템을 기능면에서 비교하고 있다.

표 4.10 Rose, Plastic과 제안된 시스템과의 비교
Table 4.10 Comparison between Rose, Plastic and proposed system

시스템 / 지원 분야	Rational Rose 2000 Enterprise	PLASTIC 3.0	제안 시스템
지원하는 다이어그램 수	9	8	1
순공학 지원	○	○	×
역공학 지원	○	○	×
데이터베이스 지원	×	×	○
실시간 정보 공유	×	×	○

제 5 장 결론 및 향후 연구과제

UML은 기존의 다양한 시스템 개발 방법론들을 하나의 틀로 통합한 것이다. 특히, 소프트웨어 시스템의 설계, 개발 등을 체계적으로 지원하는 모델링 언어이다. 이러한 UML로 개발된 모델들의 효율적인 사용을 위해서는 모델들을 통합하여 저장하고 관리할 필요성이 있다. 따라서 본 논문에서는 UML 클래스 다이어그램을 관계형 데이터베이스로 변환시키는 방법과 검색하는 방법을 제안하였다. 제안한 방법에 따라 모델링 정보를 데이터베이스화함으로써 변화하고 있는 컴퓨팅 환경과 개발 방법론에 빠르게 대처할 수 있다. 또한 다수의 사용자가 설계에 참여할 수 있도록 모델링 정보를 공유함으로써 설계 과정의 효율을 향상시키는 데도 기여할 수 있을 것이다. 현재 가장 많은 사용자를 확보하고 있는 관계형 데이터베이스를 사용함으로써 현재의 데이터베이스 시스템에 바로 적용할 수 있다는 장점도 가지게 된다.

본 연구의 향후 과제로는 현재 클래스 다이어그램에 한정된 관계형 데이터베이스로의 저장 및 검색 방법을 UML로 표기된 모든 종류의 다이어그램으로 확대하여 적용하는 것이다. 또한 본 논문에서 제안한 방법에 소프트웨어 순공학(forward engineering) 및 역공학(reverse engineering) 기법을 도입하여 모델링이 곧 프로그래밍과 연결될 수 있도록 하는 것이다. 또한 분산 환경에서의 소프트웨어 개발을 지원할 수 있도록 해야 하며, UML로 생성된 모델들의 OODB(Object Oriented Database)로의 저장 방법도 개발할 필요성이 있다.

참 고 문 헌

[1] J. Rumbaugh, *Object-Oriented Modeling and Design*, Prentice-Hall, 1991.

[2] G. Booch, "Object-Oriented Development", *IEEE Transactions on Software Engineering*, SE-12, pp. 211-221, 1986.

[3] G. Booch, *Object-Oriented Analysis and Design with Application*, 2nd ed., Benjamin Cummings Pub., 1994.

[4] I. Jacobson, *Object-Oriented Software Engineering : A Use Case Driven Approach*, Addison-Wesley, 1992.

[5] http://www.omg.org - OMG home page, specification for UML and related modeling standards, such as MOF and XMI.

[6] http://www.rational.com/uml - Rational Software's UML Resource Page.

[7] http://www.plasticsoftware.com - Plastic Software's Home Page.

[8] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 3th ed., McGraw-Hill, 1996.

[9] KMK **정보산업연구원**, ***관계형*** *DB**의 객체지향기법***, **삼각형**, 1995. 5.

[10] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley, 1997.

[11] M. Priestley, *Practical Object-Oriented Design with UML*, McGraw

-Hill, 2000.

[12] P. W. Sheridan and J. M. Sekula, *Iterative UML Development using Visual C++ 6.0*, WORDWARE, 1999.

[13] http://www.uml.co.kr - UML **및 객체지향 관련 학술**, **토론 사이트**.

[14] **왕창종**, ***소프트웨어 공학***, **正益社**, 1998.

[15] **강설문**, **김태희**, "**객체지향 소프트웨어 개발 방법론의 표준화** : UML (Unified Modeling Language)", ***정보처리논문지***, **제**5**권 제**5**호**, pp. 64-73 1998. 8.

[16] **이성대**, **박휴찬**, "UML **모델의 저장 및 질의를 위한 관계형 데이터베이스 설계 및 구현**", *2000 **추계학술발표 논문집***, **상권**, pp. 79-82, **한국정보처리학회**, 2000. 10.

[17] **이경환**, ***최신 소프트웨어 공학 사례중심***, **清文閣**, 1998.