데이터베이스에 기반한 UML 모델 관리시스템

이 성 대 · 박 휴 찬

Database Supported System for UML Models Management

Seong-Dae Lee · Hyu-Chan Park

요 약

UML(Unified Modeling Language)은 소프트웨어 설계뿐만 아니라 네트워크 상에서의 데이터 전송, 가상 데이터를 이용한 물리현상, 회로 분석 및 설계 등 다양한 형태의 시뮬레이션에서도 사용되고 있는 모델링 언어이다. 이러한 UML로 개발된 소프트웨어 설계 모델이나 시뮬레이션 모델들을 효율적으로 저장하고 관리할 수 있는 통합 시스템의 필요성이 제기되고 있다. 이를 위하여 본 논문에서는 UML 모델들의 저장과 관리의 효율성을 높이기 위해서 저장하부 구조를 데이터베이스화할 수 있는 방법을 제안한다. 제안한 방법을 적용한 시스템은 UML 모델들을 다수의 사용자들이 서로 공유할 수 있으며 모델의 재사용성을 높이고 모델 정보의 효율적인 검색을 지원할 것이다.

1. 서론

모델링·시뮬레이션은 모델을 설계하고, 설계된 모델을 실행하고, 실행 결과를 분석하는 순환적인 프로세스로 구성될 수 있다. 시뮬레이션 모델의 설계 단계에서는 다양한 방법들이 사용되고 있는 데, 최근에 많이 사용되는 것이 UML(Unified Modeling Language)이다. 이러한 UML은 소프트웨어 개발뿐만 아니라 네트워크 모델링[1], 하드웨어와 소프트웨어의 동시 설계(co-design)[2], 가스 터보 엔진 시뮬레이션[3] 등 다양한 분야에서 시뮬레이션 모델의 개발에 사용되고 있다.

UML은 OMT 방법론[4], Booch 방법론[5,6], OOSE 방법론[7] 등 다수의 객체지향 방법론들을 통합하였으며, 객체 지향 시스템의 개념적, 물리적 표현을 지원하며 소프트웨어 시스템이나 하드웨어 시스템 모두를 모델링할 수 있는 표준 모델링 언어이다. 특히 사용자와 개발자 사이의 의사 소통에 공동으로 사용할 수 있으며, 개발과 문서화 작업을 동시에 병행할 수 있게 한다. 또한 많은 컴포넌트나 에이전트들을 가진 대형 시스템의 경우 부시스템을 재사용할 수 있는 방법을 제공한다.

UML은 아홉 가지 종류의 다이어그램들로 구성되어있다. 각 다이어그램들은 시스템의 정적인 혹은 동적인 상태나 동작을 나타내며, 개발 과정에서 순환적으로 적용될 수 있다. 따라서 다양한 표기법을 사용하고 있는 UML 다이어그램들을 효율적으로 저장하고 관리할 수 있는 개발도구의 필요성이 제기되고 있으며, 많은 개발 도구들이 UML 다이어그램을 통합하여 관리할 수 있는 기능을 제공하고 있다[8,9]. 하지만 기존의 개발 도구들은 UML 모델 정보들을 파일 시스템을 사용하여 저장한다. 파일 저장시스템은 효율적인 저장과 설계 정보의 검색 그리고 여러 개발자들의 공동 설계 참여에 어려움이 있다.

* 한국해양대학교 컴퓨터공학과

따라서 본 논문에서는 UML 다이어그램을 관리할 수 있는 데이터베이스를 설계하고, 다양한 질의를 통하여 이미 생성된 다이어그램의 정보를 검색할 수 있는 방법을 제안한다. 이를 위해, 먼저 다이어그램을 구성하고 있는 구성요소들을 분류한다. 예를 들면, 클래스 다이어그램은 크게 클래스(class)와 관계(relationship)로 구성되며, 클래스는 클래스 이름(class name)과 속성(attribute), 연산(operation)으로 구성된다. 이러한 구성요소들을 먼저 분류한 후 분류된 각각의 구성요소들은 기본키(primary key)와 외래키(foreign key) 정보를 지닌 데이터베이스의 테이블로 변환한다. 제안한 방법은 다양한 질의를 통한 쉬운 검색, 다수 사용자들의 모델 정보의 공유, 모델의 재사용 등 많은 장점이 있다.

2. UML 및 관련 연구

UML은 시스템을 가시화(visualizing), 명세화(specifying), 구조화(constructing), 문서화(docu- menting)하는 모델링 언어(modeling language)이다. 특히 기존에 존재하던 여러 개발 방법론[4-7]들의 장점들을 모두 수용했으며, 확장이 용이하고, 다양한 표기법을 가지고 있어 다양한 종류의 시스템 개발에 사용되어질 수 있다.

2.1 UML의 구성 요소 및 다이어그램

UML은 크게 사물(things), 관계(relationships), 다이어그램(diagrams)의 3가지 구성요소로 구분할 수 있다. 이 중에서 모델을 시각화할 수 있는 구성요소가 다이어그램으로 다시 9가지로 세분할 수 있다. 본 논문에서는 이 중에서 클래스 다이어그램(class diagram), 사용사례 다이어그램(use case diagram), 활동 다이어그램(activity diagram)에 대하여 데이터베이스에 저장하고 관리할 수 있는 방법을 제안한다. 본 논문에서 다루지 않은 다이어그램들에게도 동일한 방법을 적용할 수 있다.

클래스 다이어그램은 내부에 클래스와 관계로 구성이 되고, 클래스는 클래스 이름과 속성 목록, 연산 목록으로 구성이 된다. 인터페이스는 연산들의 모임으로 하나의 클래스나 컴포넌트들이 제공하며, 서비스를 명세화하기 위해 사용한다[10,11].

사용사례 다이어그램은 다수의 사용사례(use case)들 사이의 관계를 도식화하며, 사용사례는 전체 시스템에서 일부 기능을 나타내며, 사용사례, 행위자(actor), 시스템(system)과 같은 구성요소와 각 구성요소들 사이의 관계로 구성된다. 사용사례 다이어그램에서 사용되는 관계는 행위자와 사용사례 사이의 연관관계, 행위자들 간의 일반화 관계, 사용사례들 사이의 일반화, 확장(extend), 포함(include)이 있다[10,11].

활동 다이어그램은 순서도나 병렬적인 처리를 필요로 하는 행위를 표현할 때 사용한다. 이 다이어그램의 목적은 내부적인 처리에 의해서 구동되는 흐름을 표현하는 것이다. 구성요소로는 상태를 나타내는 활동(activity), 활동을 분해한 동작(action), 시작(start), 멈춤(stop), 조건 분기를 나타내는 결정 심벌(decision symbol), 클래스 다이어그램의 인스턴스인 객체, 신호 또는 메시지들의 전송과 수신 그리고 흐름, 기능을 고려하여 활동들을 그룹화한 구획면(swimlane) 등이 있다[10,11].

2.2 관련 연구

현재 UML에 대한 연구는 국내외적으로 활발히 진행되고 있다. 대표적인 연구 분야로는 UML 표기법으로 작성한 모델을 XML로 변환하는 연구로서 [12,13]에서는 DOM(Document Object Model)과 CORBA (Common Object Request Broker Architecture)를 이용하여 모델을 XML 문서로 변환하는 UXF (UML

eXchange Format)를 제안하였다. [14]에서는 UXF를 확장하여 UDXF(UML Data eXchange Format)를 제안하였다.

[15]에서는 요구 분석 단계에서 사용되는 사용 사례 다이어그램을 정형화시키는 방법을 제안하였으며, [16]에서는 소프트웨어 개발 과정에서 UML을 사용하는 사용자 중심의 개발 방안으로 DYNAMITE(DYNAMIc Task nEts) 모델을 제안하였다.

UML을 지원하는 대표적인 상용 시스템은 국외의 "Rose"[8]와 국내의 "Plastic"[9]이 있다. 이들 시스템에서 모델 정보를 저장하는 방법은 파일 저장 시스템을 이용한다. UML의 모든 다이어그램을 지원하며, 다이어그램을 C++, Java 등과 같은 객체 지향 언어의 코드를 생성시키는 순공학(forward enginee- ring)과 객체를 표현하고 있는 코드에서 다이어그램을 생성하는 역공학(reverse engineering) 기능을 지원한다. 이들 시스템은 각각 고유한 파일 형태로 모델 정보를 저장하기 때문에 개발 도구 사이에서 호환성을 가질 수 없다는 단점이 있다.

이렇듯 많은 연구들이 수행되고 있고, 많은 개발 도구들이 UML을 지원하고 있지만 모델을 데이터베이스에 저장하여 공동으로 작업을 수행할 수 있게 하는 연구는 미흡하다.

3. UML 다이어그램 관리를 위한 데이터베이스 설계

UML의 표기법을 사용한 모델링 정보는 다수의 다이어그램들로 구성된다. 우선 전체 다이어그램들을 관리할 수 있는 다이어그램 관리 테이블이 필요하며 스키마는 그림 1과 같다. 그림 1의 DIAGRAM 테이블에서 DID는 이후 생성될 클래스, 사용사례, 활동 다이어그램에서 참조하게된다.

```
create table DIAGRAM
(
    DID int primary key,
    NAME varchar not null,
    INFO varchar,
    OWNER char not null,
    DATE datetime,
    ALLOW boolean not null
);
```

그림 1. 다이어그램 정보 저장 스키마

3.1 UML 클래스 다이어그램 변환

클래스 다이어그램은 다수의 클래스와 관계들로 구성된다. 각 클래스 내부에는 클래스의 이름과 다수의 속성들, 연산들이 존재하는 구조를 지니고 있으며 그림 2와 같은 구조를 지니고 있다[10]. 인터페이스는 그림 2(a)와 같이 원을 사용하는 방법과 그림 2(b)와 같이 클래스의 스테레오타입(stereotype)에서 인터페이스를 명시하는 방법이 있다.

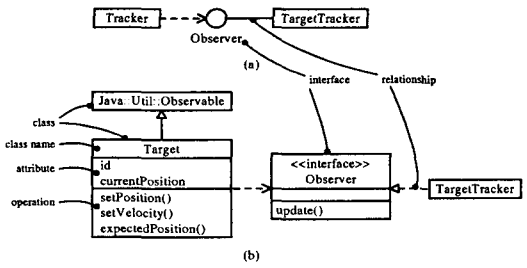

그림 2. 클래스 다이어그램 예제

클래스 다이어그램을 구성하고 있는 클래스와 관계를 저장할 수 있는 스키마는 그림 3과 같다. CLASS 테이블의 DID는 DIAGRAM 테이블의 DID를 참조하는 외래키이다. 또한 화면 출력을 위해 클래스의 시작 좌표(START_X, START_Y)를 저장할 수 있도록 고려하였다. CLASS_ RELATIONSHIP 테이블은 클래스나 인터페이스와 같은 모델 요소간의 관계를 저장하는 테이블로써 다중성(multiplicity)과 역할명(role name)도 저장할 수 있도록 해야한다.

```
create table CLASS
(
    CID int primary key,
    DID int foreign key references DIAGRAM(DID),
    NAME varchar not null,
    TAG varchar,
    IS_INTERFACE bool,
    STEREOTYPE varchar,
    START_X int not null,
    START_Y int not null,
    INFO varchar not null
);

create table CLASS_RELATIONSHIP
(
    RID int primary key,
    SUPER_CID int foreign key references CLASS(CID),
    SUB_CID int foreign key references CLASS(CID),
    TYPE int not null,
    STEREOTYPE varchar,
    SUPER_MULTI varchar,
    SUB_MULTI varchar,
    SUPER_ROLE varchar,
    SUB_ROLE varchar,
    INFO varchar
);
```

그림 3. 클래스 및 관계 저장 스키마

클래스 내부의 구성요소로는 속성과 연산이 있으며, 속성과 연산은 공통적으로 타입과 가시성(visibility) 표시를 가진다. 속성과 연산을 저장할 수 있는 관계형 스키마는 그림 4와 같다.

```
create table ATTRIBUTE
(
    AID int primary key,
    CID int foreign key references CLASS(CID),
    NAME varchar not null,
    VISIBILITY int not null,
    TYPE varchar,
    STEREOTYPE varchar,
    INFO varchar
);

create table OPERATION
(
    OID int primary key,
    CID int foreign key references CLASS(CID),
    NAME varchar not null,
    PARAMETER varchar not null,
    VISIBILITY int not null,
    TYPE varchar not null,
    STEREOTYPE varchar,
    INFO varchar
);
```

그림 4. 속성 및 연산 저장 스키마

그림 4에서 속성 목록은 ATTRIBUTE 테이블에, 연산 목록은 OPERATION 테이블에 저장한다. 두 테이블에 저장되는 모든 값들은 CLASS 테이블의 기본키인 CID를 참조하여야 한다. 추가적으로 연산의 반환 타입(return type), 인자(parameter)에 대한 정의를 추가하였다.

3.2 사용사례 다이어그램 변환

사용 사례 다이어그램은 행위자(actor), 시스템(system), 사용사례(use case) 등과 같은 구성요소와 각 구성요소간의 관계로 구성된다. 그림 5는 간단한 사용사례 다이어그램이다 [17].

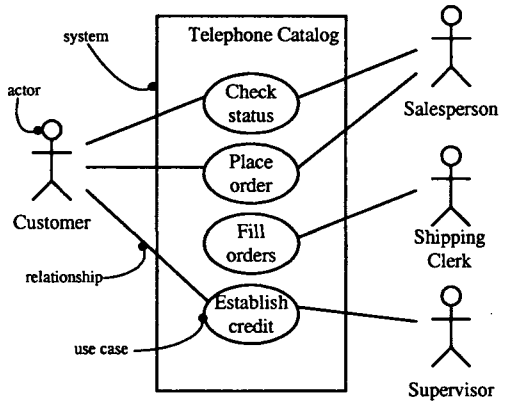

그림 5 사용사례 다이어그램 예제

사용사례 다이어그램의 구성요소와 관계를 저장할 수 있는 스키마는 그림 6과 같다.

```
create table USE_CASE
(
    UID int primary key,
    DID int foreign key references DIAGRAM(DID),
    NAME varchar not null,
    TYPE int not null,
    STEREOTYPE varchar,
    START_X int,
    START_Y int,
    INFO varchar
);

create table USE_CASE_RELATIONSHIP
(
    RID int primary key,
    SUPER_UID int foreign key references USE_CASE(UID),
    SUB_UID int foreign key references USE_CASE(UID),
    TYPE int not null,
    STEREOTYPE varchar,
    INFO varchar
);
```

그림 6. 사용사례 다이어그램 저장 스키마

USE_CASE 테이블에 있는 DID는 DIAGRAM 테이블을 참조하며, TYPE의 값에 따라 행위

자, 시스템, 사용사례 등의 구성요소들을 구분한다. USE_RELATIONSHIP 테이블은 사용사례 다이어그램 내부의 구성요소들 사이의 관계를 저장하는 테이블로서 SUPER_UID는 관계를 시작하는 구성요소를 나타내고, SUB_UID는 관계가 끝나는 구성요소를 나타내며, 각각은 USE_CASE 테이블의 UID를 참조한다.

3.3 활동 다이어그램 변환

활동 다이어그램은 활동(activity), 액션(action), 객체(object), 구획면(swimlane) 등의 구성요소와 구성요소간의 관계로 구조를 표시할 수 있다. 그림 7은 소프트웨어 개발 프로세스(software development process)를 활동 다이어그램으로 모델링한 것이다[17].

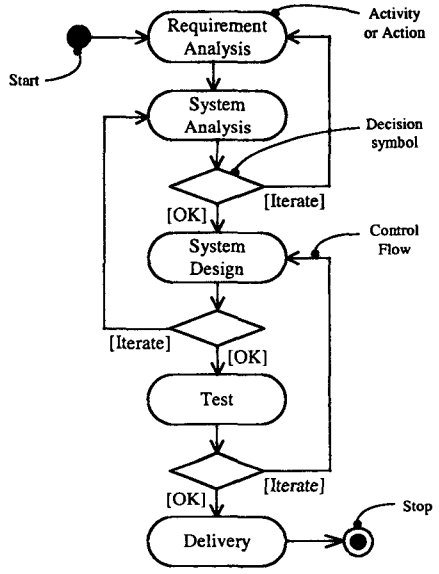

그림 7. 활동 다이어그램 예제

그림 8은 활동 다이어그램의 정보를 저장할 수 있는 관계형 데이터베이스 스키마로써 각 구성요소들은 ACTIVITY 테이블의 TYPE 값에 따라 구별되며, 흐름은 ACTIVITY_RELATIONSHIP 테이블의 TYPE 값에 따라 제어 흐름과 객체 흐름으로 구분하도록 한다.

```
create table ACTIVITY
(
        AID int primary key,
        DID int foreign key references DIAGRAM(DID),
        NAME varchar not null,
        TYPE int not null,
        STEREOTYPE varchar,
        INFO varchar,
        START_X int,
        START_Y int
);

create table ACTIVITY_RELATIONSHIP
(
        RID int primary key,
        SUPER_AID int foreign key references ACTIVITY(AID),
        SUB_AID int foreign key references ACTIVITY(AID),
        TYPE int not null,
        STEREOTYPE varchar,
        INFO varchar
);
```

그림 8. 활동 다이어그램 저장 스키마

4. 시스템 구현

본 장에서는 제안한 방법을 적용하여 구현한 시스템의 전반적인 흐름과 다이어그램의 검색 방법, 구현한 시스템을 사용하여 작성한 예제 다이어그램을 살펴보도록 한다.

4.1 시스템 흐름도

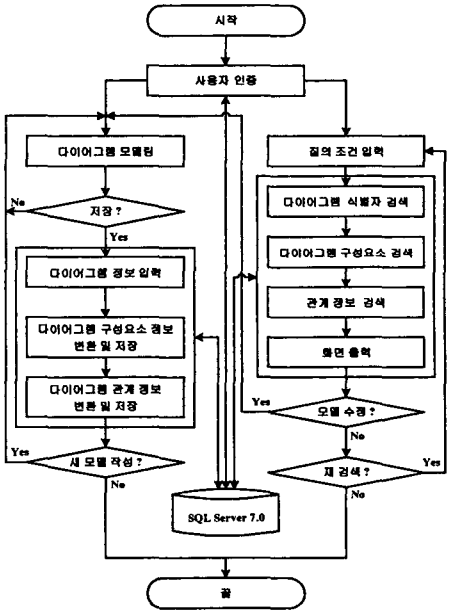

그림 10. 다이어그램 저장 및 검색 흐름도

그림 10은 본 논문에서 제안한 방법을 적용하여 구현한 시스템의 흐름도이다. 클래스, 사용사례, 활동 다이어그램의 저장 및 검색은 동일한 흐름을 가진다. 사용자 인증 과정은 시스템의 보안을 위해서 필요하다. 저장 루틴에서는 먼저 다이어그램의 정보를 입력받고, 이 후 작성한 다이어그램의 구성요소를 분류하여 각 테이블에 저장할 수 있도록 변환한다. 변환된 구성요소 정보는 각 테이블에 저장되며, 이 후 각 구성요소 사이의 관계 정보가 저장이 되도록 한다. 검색 루틴에서는 먼저 검색어가 입력이 되면 검색어를 기반으로 다이어그램의 식별자를 찾는다. 검색 조건에 맞는 다이어그램 식별자가 찾아지면 다이어그램의 내부에 존재하는 구성요소들을 먼저 찾고 구성요소에 따라 관계 정보를 검색한다. 검색 과정이 끝나면 검색된 구성요소 정보와 관계 정보를 이용하여 화면에 출력하도록 하며 모델이 수정되면 다이어그램 저장 루틴을 수행한다.

4.2 UML 다이어그램 검색

임의의 다이어그램을 검색하기 위해서는 사용자가 입력하는 조건을 만족하는 다이어그램의 식별자, DID를 먼저 검색해야한다. DID가 결정되면 테이블의 관계를 이용하여 각 다이어그램 내부의 구성요소들을 차례로 검색할 수 있다.

```
조건 : 클래스 이름이 'display'인 클래스를 포함하고 있는
       다이어그램을 찾으시오.
검색 과정

// 다이어그램 식별자 검색
step 1 : SELECT DID as $did FROM CLASS
                 WHERE CLASS_NAME = 'display';

// 다이어그램 정보 검색
step 2 : SELECT * FROM DIAGRAM
                 WHERE DID = $did;

// 소속 클래스 식별자 검색
step 3 : SELECT CID as $cid FROM CLASS
                 WHERE DID = $did;

// 각 클래스 내부의 속성, 연산, 관계 검색
for all $cid
     step 4 : SELECT * FROM ATTRIBUTE
                      WHERE CID = $cid;
     step 5 : SELECT * FROM OPERATION
                      WHERE CID = $cid;
     step 6 : SELECT * FROM CLASS_RELATIONSHIP
                      WHERE SUPER_CID = $cid
                       OR SUB_CID = $cid;
end for
```

그림 11. 클래스 다이어그램 검색 구문

클래스 다이어그램의 경우, DIAGRAM 테이블의 DID가 결정되면 CLASS 테이블에서 DID의 값을 이용하여 모든 클래스의 식별자 CID를 먼저 추출한다. 추출된 CID를 이용하면 각 클래스 내부의 속성들과 연산들을 ATTRIBUTE, OPERA- TION 테이블로부터 찾을 수 있다. 마지막으로, 각 클래스 및 인터페이스간의 관계들을CLASS_ RELATIONSHIP 테이블에서 검색한다. 사용사례와 활동 다이어그램의 경우, DID가 결정되면 UID 또는 AID를 찾고 그 값에 따라 각각의 관계 테이블로부터 RID를 검색한다. 그림 11은 클래스 다이어그램을 검색하기 위해서 클래스 이름이 조건으로 주어지는 경우 다이어그램을 검색하는 과정을 SQL 구문으로 간략하게 기술한 것이다.

4.3 시스템 구현

본 논문에서 제안한 방법을 구현하기 위하여 관계형 데이터베이스로는 Microsoft SQL Server 7.0을 사용하였고, 프로그래밍 언어로는 Visual C++ 6.0을 사용하였다. 그림 12는 본 논문에서 제안한 방법을 적용하여 구현한 UML 다이어그램 관리 시스템의 구조도이다. 사용자가 모델을 직접 작성할 수 있는 다이어그램 편집기와 작성된 다이어그램에서 구성요소와 관계 정보를 추출하여 저장하는 다이어그램 정보 저장 모듈, 각 구성요소와 관계를 검색하여 화면 출력하는 다이어그램 검색 모듈로 구성되며 데이터베이스와 연동하도록 구현하였다.

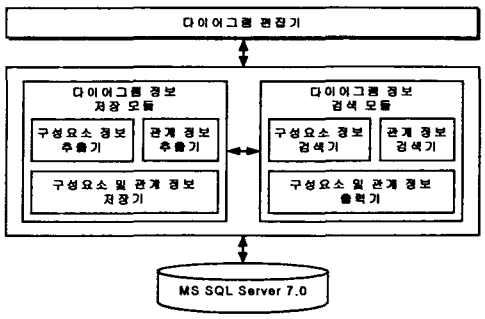

그림 12. 다이어그램 관리 시스템 구조도

그림 13은 본 논문에서 제안한 방법을 적용하여 구현한 시스템을 사용하여 모델링한 무인 운반 차량의 기본적인 운행을 모니터링하는 예제 클래스 다이어그램이다. 클래스 "Sensor"의 하위 클래스로는 장애물을 탐지하는 클래스 "Hurdle_ Sensor"와 운반경로를 탐지하는 클래스 "Line_ Sensor"가 있다. 이 두 클래스는 "Controller" 클래스와 연관 관계를 지니며 장애물과 경로를 탐지하여 클래스 "Controller"로 탐지 내용을 보내는 역할을 하게 된다. 그리고 "Controller" 클래스는 각각의 차량의 ID에 따라서 센서 클래스를 제어해야 한다. 클래스 "Controller"는 클래스 "Motor", "Display", "Monitoring"과도 연관 관계를 지니고 있다. "Controller"와 "Monitoring" 클래스와의 관계에서 각각의 다중성(multiplicity)은 "1..*", "1"이다. 즉, 현재 하나 이상의 무인 운반 차량을 감시하는 하나의 시스템이 존재한다는 것을 의미한다. "Controller" 클래스 내부에는 현재 위치 정보, 목적지 위치 정보, 장애물 발견 유무, 현재 속도 등을 나타내는 속성과 센서를 통하여 장애물 탐지, 경로 탐지, 차량을 운영하는 연산들이 필요하다. "Monitoring" 클래스에서는 속성으로 화면에 출력할 차량의 ID와 현재 상태 및 위치를 파악하는 연산들로 구성된다.

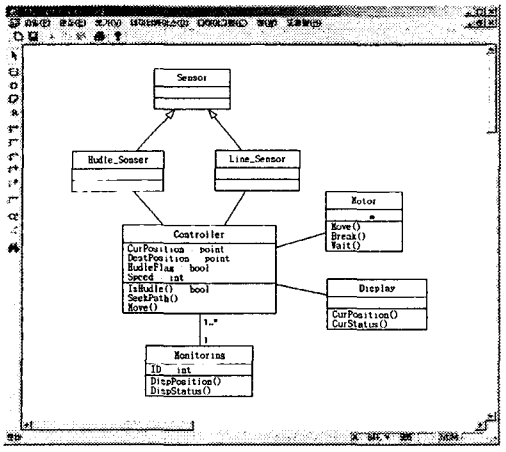

그림 13. 구현한 시스템을 사용한 예제 클래스 다이어그램

5. 결론 및 추후 연구과제

본 논문에서는 UML 다이어그램을 저장하고 관리할 수 있는 관계형 데이터베이스를 설계하였고, 검색하는 방법을 제안하였다. 먼저 클래스, 사용사례, 활동 다이어그램을 구성하고 있는 각각의 요소들을 분류하여 테이블로 구성하고 관계형 데이터베이스의 기본키와 외래키 정보를 이용하여 각 테이블의 관계를 유지하는 방법을 사용하였다. 제안한 방법에 따라 모델링 정보를 데이터베이스화함으로써 다수의 개발자가 모델링 정보를 공유하여 설계에 공동으로 참여할 수 있음으로 설계의 효율성을 향상시킬 수 있으며, 이미 작성된 모델링 정보를 쉽게 검색하여 재사용함으로 비용 절감의 효과도 기대할 수 있다. 또한 현재 가장 많은 사용자와 개발자를 확보하고 있는 관계형 데이터베이스를 사용함으로써 현재의 데이터베이스 시스템에 바로 적용할 수 있다는 장점도 가지게 된다.

추후 연구과제로는 설계와 구현을 동시에 가능하도록 하기 위해 순공학(forward engineering) 및 역공학(reverse engineering) 기법을 도입하여 모델링이 곧 프로그래밍과 연결될 수 있도록 하는 것이다. 또한 객체 관계형(object relational) 및 객체 지향(object oriented) 데이터베이스에서도 UML 다이어그램 정보를 저장 및 검색할 수 있도록 해야할 것이다.

참고문헌

1) 이종혁, 박성공, 김영준, 백두권, "NGN 콜 에이전트의 속성기반 분석과 객체 지향 설계", '00춘계학술대회 논문집, pp.84-90, 한국시뮬레이션학회, 2000.

2) G. Martin, L. Lavagno and J. L. Guerin, "Embedded UML: a merger of real-time

UML and co-design", *Proc. of the 9th Int. Symposium on Hardware/Software Codesign*, pp.23-28, Apr. 2001.

3) Z. Xie, J. Yu and J. Liu, "*Applying UML to Gas Turbine Engine Simulation*", Proc. of TOOLS 31, pp.458-464, IEEE, Sep. 1999.
4) J. Rumbaugh, *Object-Oriented Modeling and Design*, Prentice-Hall, 1991.
5) G. Booch, "*Object-Oriented Development*", IEEE Transactions on Software Engineer- ing, SE-12, pp. 211-221, 1986.
6) G. Booch, *Object-Oriented Analysis and Design with Application*, 2nd ed., Benjamin Cummings Pub., 1994.
7) I. Jacobson, *Object-Oriented Software Engineering : A Use Case Driven Approach*, Addison-Wesley, 1992.
8) Rational Software's UML Resource Page, http://www.rational.com/uml
9) Plastic Software's Home Page, http:// www.plasticsoftware.com
10) OMG UML Specification v. 1.4 draft, http://www.omg.org/
11) G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley, 1997.
12) J. Suzuki and Y. Yamamoto, "*Toward the interoperable software design models : quartet of UML, XML, DOM and CORBA*", ISESS'99, 1999.
13) J. Suzuki and Y. Yamamoto, "*Managing the software design documents with XML*", Proc. of the 16th Annual Int. Conf. on Comput. Documentation, Sep. 1998.
14) 최동운, 김진성, 송행숙, "XML DTD를 이용한 UML 설계정보의 변환 규칙", 2000 추계학술발표 논문집, pp.71-74, 한국정보처리학회, 2000.
15) X. Li, Z. Liu and J. He, "*Formal and Use-Case Driven Requirement Analysis in UML*", Proc. of the 25th Conf. of AICSA, pp.215-224, Oct. 2001.
16) D. Jager, A. Sechleicher and B. Westfechtel, "*Using UML for Software Process Modeling*", Proc. of the 7th ACM SIGSOFT Symposium, Vol.24, No.6, pp.91-108, Oct. 1999.
17) H. Eriksson and M.Penker, *Business Modeling with UML*, OMG Press, 2000.
18) 이성대, 박휴찬, "UML 모델의 저장 및 질의를 위한 관계형 데이터베이스 설계 및 구현", 2000 추계학술발표 논문집, pp.79-82, 한국정보처리학회, 2000.