요구분석을 위한 UML 다이어그램 저장관리 시스템

(UML Diagrams Repository System for Requirement Analysis)

이 성 대 † 박 휴 찬 ††
(Seong-Dae Lee) (Hyu-Chan Park)

요 약 소프트웨어 생명주기는 요구분석, 설계, 구현, 유지보수 단계 등으로 구성되며, 각 단계의 산출물은 다양한 형태를 지니고 있다. 이러한 산출물을 표준화한 것이 UML이며, 요구분석 단계에서는 클래스 다이어그램, 사용사례 다이어그램, 활동 다이어그램, 협력 다이어그램이 일반적으로 사용되고 있다. 기존의 UML 개발 도구들은 이러한 다이어그램을 파일에 저장하기 때문에, 다이어그램 정보를 검색하거나 다수의 개발자들이 참여하는 공동 작업시 어려움이 발생한다.

본 논문에서는 이러한 어려움을 해결하고 요구분석 단계에서의 산출물을 효율적으로 처리하기 위하여 다이어그램 정보를 데이타베이스에 저장하고 관리하는 방법을 제안한다. 이를 위해 클래스, 사용사례, 활동, 협력 다이어그램을 구성하고 있는 모든 요소들을 분류하고, 분류된 각 구성요소들을 관계형 테이블로 변환한다. 이렇게 변환된 테이블에 다이어그램 정보를 저장하고 다양한 형태의 질의를 통해 저장된 정보를 검색할 수 있도록 한다. 제안한 방법은 다이어그램 정보를 다수의 개발자가 공유하여 사용할 수 있으며 모델의 재사용에 편리성을 제공할 것이다.

키워드 : UML, 데이타베이스, 요구분석

Abstract Software life cycle consists of requirement analysis, design, implementation, and maintenance phases, and the product of each phase has various format. The UML normalizes such products, and the class diagram, use case diagram, activity diagram and collaboration diagram are usually used for the requirement analysis phase. Because most of UML development tools store such diagrams in a file, there may be some difficulties of information retrieval and co-work among users.

To cope with the difficulties, this paper proposes a database supported methodology to store and manage the diagrams produced by the requirement analysis. In this methodology, the constituents of class, use case, activity and collaboration diagram are first analyzed and then transformed in the form of relational tables. The constituents of such diagrams are stored as tables in a database, and can be easily retrieved from the database by using some queries. This database supported methodology provides the concurrent sharing and high reuse of diagrams.

Key words : UML, Database, Requirement Analysis

1. 서 론

최근 소프트웨어가 고부가가치 상품으로 인식되고 있고, 업무 처리 과정에서 소프트웨어가 처리할 업무의 범위와 규모가 커짐에 따라 시스템의 복잡성을 체계적으로 관리할 필요성이 증가되고 있다. 따라서 소프트웨어의 생산을 자동화하고 개발에 필요한 시간과 비용을 절감하여 소프트웨어의 질을 향상시킬 수 있는 기술들이 모색되고 있는 데, 이 중에서 개발할 시스템의 구성 요소를 체계적으로 분석하고 설계할 필요성에 따라 제안된 것이 UML(Unified Modeling Language)이다.

UML은 소프트웨어 개발을 위한 표준 모델링 언어로

· 이 논문은 2002년도 두뇌한국21사업에 의하여 지원되었음.
† 학생회원 : 한국해양대학교 컴퓨터공학과
omega@hhu.ac.kr
†† 정 회 원 : 한국해양대학교 기계·정보공학부 교수
hcpark@hhu.ac.kr
논문접수 : 2002년 4월 23일
심사완료 : 2002년 8월 20일

빠르게 자리잡고 있다. UML은 Rumbaugh의 OMT 방법론[1], Booch의 Booch 방법론[2, 3], Jacobson의 OOSE 방법론[4] 등 다수의 객체지향 방법론들을 통합하여 만든 통합 모델링 언어이다. 현재 일부 시스템 개발 도구들이 UML을 지원하고 있고[5, 6], 또한 많은 소프트웨어 시스템을 개발하는 과정에서 UML을 사용하고 있다[7, 8]. UML은 아홉 가지 종류의 다이어그램들로 구성되어있다. 각 다이어그램들은 시스템의 정적 혹은 동적인 상태나 동작을 나타내며, 소프트웨어의 개발 과정에서 생산되는 다양한 종류의 산출물에 이용되고 있다[9]. 따라서 다양한 종류의 산출물을 효율적으로 저장하고 관리할 수 있는 개발도구의 필요성이 제기되고 있으며, 이를 위해 시스템 개발 도구들은 소프트웨어 개발과정에서 나오는 UML 다이어그램을 통합하여 관리할 수 있는 기능을 제공하고 있다[5, 6]. 하지만 기존의 개발 도구들은 UML의 표기법을 이용한 설계 정보들을 저장하기 위해 특정한 형태를 지닌 파일 시스템을 사용하고 있다. 이러한 파일 저장 시스템을 사용함으로 인해 효율적인 저장과 설계 정보의 검색에 어려움이 있으며, 또한 여러 개발자들이 공동으로 설계에 참여할 수 있는 방법 역시 제한되어 있다.

따라서 본 논문에서는 소프트웨어 생명주기의 요구분석 단계에서 일반적인 산출물이 되는 클래스 다이어그램(class diagram)과 사용사례 다이어그램(use case diagram), 활동 다이어그램(activity diagram) 그리고 협력 다이어그램(collaboration diagram)을 저장하고 관리할 수 있는 데이타베이스를 설계하고, 다양한 질의를 통하여 이미 생성된 다이어그램의 정보를 검색할 수 있는 방법을 제시하고자 한다. 각 다이어그램을 저장할 수 있는 데이타베이스를 설계하기 위해서 먼저 다이어그램을 구성하고 있는 구성요소들을 분류한다. 예를 들면, 클래스 다이어그램은 크게 클래스(class)와 관계(relationship)로 구성되며, 클래스는 클래스 이름과 속성(attribute), 연산(operation)으로 구성된다. 이러한 구성요소들을 먼저 분류한 후 분류된 각각의 구성요소들은 기본키(primary key)와 외래키(foreign key) 정보를 지닌 데이타베이스의 테이블로 변환하고, 테이블 사이의 관계를 설정하는 방법을 사용하여 데이타베이스를 설계한다. 설계된 데이타베이스에 저장된 다이어그램 정보는 사용자의 다양한 질의를 통해 쉽게 검색이 가능하므로 이전에 작성한 설계 정보의 재사용이 용이하고, 다수의 사용자들이 설계 정보를 실시간으로 공유할 수 있어 공동의 개발에 효율성을 더할 수 있다는 장점을 가지고 있다.

본 논문의 2장에서는 UML의 구성요소와 클래스, 사용 사례, 활동 그리고 협력 다이어그램에 대해 간략하게 살펴보고, 3장에서는 각 다이어그램을 저장하고 검색할 수 있는 관계형 데이타베이스를 설계하고, 4장에서는 제안한 방법을 적용하여 구현한 시스템의 구조와 전체적인 시스템의 흐름도에 대해서 설명하고 기존 시스템과 비교한다. 5장에서는 결론과 함께 추후 연구과제에 대해서 논의한다.

2. UML 및 관련 연구

UML은 통합된 시스템 개발방법론으로써 시스템을 가시화(visualizing), 명세화(specifying), 구조화(constructing), 문서화(documenting)하는 모델링 언어(modeling language)이다. 특히 기존에 존재하던 여러 개발 방법론[1-4]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 그림 1과 같이 크게 사물(Things), 관계(Relationships), 다이어그램(Diagrams)의 3가지 구성요소로 구분할 수 있다. 첫 번째 구성요소인 사물은 UML 모델의 실체를 나타내는 구조 사물(Structural Things), 동적인 부분을 나타내는 행동 사물(Behavioral Things), 모델의 조직적인 부분을 나타내는 그룹 사물(Grouping Things), 모델에 대한 추가적인 설명을 표현하는 주해 사물(Annotation Things)로 나눌 수 있다[9-11].

두 번째 구성요소는 관계로서 의존(Dependency), 연관(Association), 일반화(Generalization), 실제화(Realization), 집합연관(Aggregation) 관계로 나눌 수 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. 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. 일반화 관계는 "is-a" 또는 "kind-of"의 관계로서 상위 클래스의 속성(attribute)과 연산(operation)이 하위 클래스로 상속되는 것을 의미한다. 실제화 관계는 인터페이스나 컴포넌트(component)와의 상호 연관성을 표시하는 관계이다. 집합연관 관계는 연관 관계의 확장형태로써 "part-of"의 관계이다[9-11].

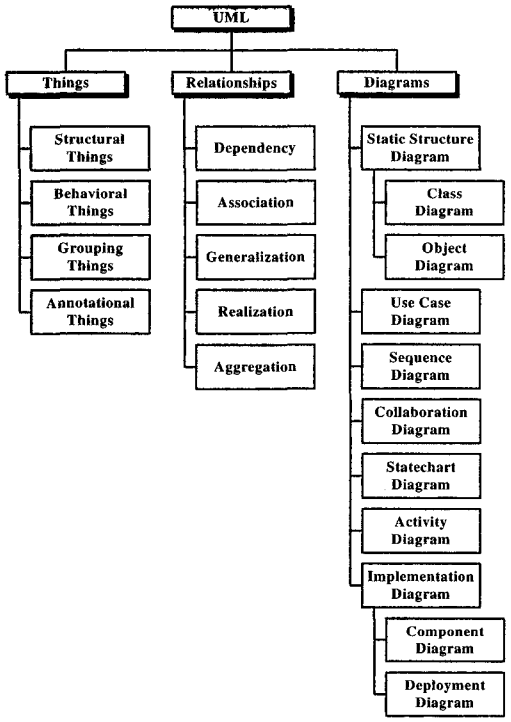

그림 1 UML의 구성 요소

세 번째 구성요소로는 설계를 시각화하는 다이어그램이다. 다이어그램을 세분화하면 정적 구조(Static Structure), 컴퓨터 시스템과 사용자의 상호작용을 표현하는 사용 사례(Use Case), 객체집합 사이에서 보내지는 메시지를 나타내는 순차(Sequence), 객체집합 사이에서 협력 관계를 표시하는 협력(Collaboration), 클래스나 시스템의 가능한 상태를 기술하는 상태도표(Statechart), 시스템에서 발생하는 활동과 작용을 표현하는 활동(Activity), 구현(Implementation) 다이어그램이 있다. 정적 구조 다이어그램은 시스템을 논리적인 관점에서 분석하는 것으로 여러 가지 객체들의 타입인 클래스(Class), 클래스의 인스턴스인 객체(Object) 다이어그램으로 세분되고, 구현 다이어그램은 소프트웨어 시스템 내부의 컴포넌트를 묘사하는데 사용되는 클래스 다이어그램의 특수한 형태인 컴포넌트(Component), 소프트웨어 시스템에서 하드웨어를 묘사하는 데 사용되는 배치(Deployment) 다이어그램으로 세분할 수 있다[9-11].

본 논문에서 다루게 될 클래스 다이어그램, 사용사례 다이어그램, 활동 다이어그램, 협력 다이어그램은 다음과 같은 구조적인 특징이 있다. 클래스 다이어그램은 내부에 클래스와 관계로 구성이 되고, 클래스는 클래스 이름(class name)과 속성 목록(attribute list) 그리고 연산 목록(operation list)으로 구성이 된다. 각각의 속성이나 연산은 가시성(visibility) 표시를 가지는 데 '+'는 public, '-'는 private, '#'은 protected 속성을 나타낸다. 클래스는 비슷한 속성과 공통의 행위를 갖는 객체들의 그룹을 기술하는 단위로써 사물을 추상화하고 단순화하는 것으로 사각형을 이용하여 표현한다. 인터페이스는 연산(operation)들의 모임으로 하나의 클래스나 컴포넌트들이 제공하며, 서비스를 명세화하기 위해 사용한다[9-11].

사용사례 다이어그램은 다수의 사용사례들 사이의 관계를 도식화하며, 일반적으로 평면 텍스트에서 정의되는 각 사용사례(use case)는 전체 시스템에서 일부 기능을 나타내며, 사용사례(use case), 행위자(actor), 시스템(system)과 같은 구성요소와 각 구성요소들 사이의 관계로 구성된다. 사용사례 다이어그램에서 사용되는 관계는 행위자와 사용사례 사이의 연관 관계, 행위자들 간의 일반화 관계, 사용사례들 사이의 일반화, 확장(extend), 포함(include)이 있다[9-11].

활동 다이어그램은 순서도나 병렬적인 처리를 필요로 하는 행위를 표현할 때 사용한다. 이 다이어그램의 목적은 내부적인 처리에 의해서 구동되는 흐름을 표현하는 것이다. 대부분의 이벤트가 내부적으로 생성된 제어의 흐름과 같은 동작을 표현해야 하는 상황에서 많이 사용하는 다이어그램이다. 활동 다이어그램은 상태를 나타내는 활동(activity), 활동을 분해한 동작(action), 시작(start), 멈춤(stop), 조건 분기를 나타내는 결정 심벌(decision symbol), 클래스 다이어그램의 인스턴스인 객체(object), 신호 또는 메시지들의 전송(send)과 수신(receive) 그리고 흐름(flow)을 나타내는 기호로 구성된다. 흐름에는 제어 흐름(control flow)과 객체 흐름(object flow)이 있다. 제어 흐름은 활동과 액션이 트랜잭션(transaction)을 통해서 연결되며, 이때의 트랜잭션을 제어 흐름이라고 한다. 하나의 제어 흐름은 동기화 형태로 여러 개의 흐름으로 분리(forking)될 수 있으며, 분리된 흐름은 다시 합쳐(joining)질 수 있다. 객체 흐름은 활동과 객체 사이의 흐름을 의미한다. 추가적으로 활동 다이어그램은 기능을 고려하여 활동들을 그룹화하는 데 이것을 구획면(swimlane)이라고 하며, 수행되는 장소를 명확하게 표현하기 위해 사용된다[9-11].

클래스 다이어그램은 시스템의 정적인 구조를 표현하지만 협력 다이어그램은 순차 다이어그램과 함께 시스

템의 동적인 구조, 즉 객체와 객체 그룹, 객체와 객체, 객체 그룹과 객체 그룹들간의 동적인 구조를 표현하는 데 사용한다. 순차 다이어그램은 시간의 흐름에 따라 행위의 흐름을 나타내는 데 중심을 두며, 협력 다이어그램은 객체간의 상호 관계에 중심을 두고 시스템을 기술하는 다이어그램이다. 또한 협력 다이어그램은 사용 사례 내부에 포함된 객체들 사이의 행위를 보일 때 필요하다. 주요 구성요소로는 객체, 메시지(message), 링크(link), 메시지의 순서(sequence) 등이 있다.

2.2 관련 연구

현재 UML에 대한 연구는 국내외적으로 활발히 진행되고 있다. 대표적인 연구 분야로는 UML 표기법으로 작성한 모델을 XML(eXtensible Markup Language)로 변환하는 연구로서, 많은 개발도구들 사이에서 UML 모델 정보를 교환하기 위해 현재 웹에서 문서 표준으로 자리잡고 있는 XML을 이용하려는 것이다. [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. UXF에서는 UML의 의미정보를 XML의 DTD(Document Type Definition)를 이용하여 XML 파일로 변환한 후 웹 상에서 개발자들이 공유하여 개발에 공동으로 참여할 수 있도록 하였다.

UML과 관련된 다른 연구로 [14]에서는 관계형 데이타베이스를 설계하기 위해서 UML과 OCL(Object Constraint Language)를 활용하는 방안을 제시하였다. 이 논문에서는 데이타베이스 응용 프로그램을 작성할 때, 프로그램에서 제약 사항을 검토하는 것이 아니라 테이블을 생성하는 시점에서 제약사항을 명시하는 방법으로 UML과 OCL을 활용하였다. 따라서 UML 클래스 다이어그램을 활용하여 관계형 데이타베이스 스키마를 생성하는 방법으로 각 클래스마다 하나의 테이블을 생성하고 기본키와 외래키 정보를 이용하여 테이블간의 관계가 클래스들간의 관계로 결정하는 방법을 사용하고 있다.

이외에도 [15]에서는 요구분석 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 UML을 제안한 Booch, Rambaugh와 Jacobson이 중심이 되어 개발한 "Rose"와 국내의 "Plastic"이 있다. 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[5]. Plastic은 국내에서는 UML을 지원하는 대표적인 도구로써 모델 정보의 저장은 파일 시스템을 사용하며 모든 종류의 다이어그램을 지원하고 있다. 또한 자바(java) 코드에 대한 순공학과 역공학을 지원한다[6]. 위의 두 개발 도구는 각각 고유한 파일 형태로 모델 정보를 저장함으로 두 개발 도구 사이에서는 호환성을 가질 수 없다는 단점이 있다.

이렇듯 많은 연구들이 수행되고 있고, 많은 개발 도구들이 UML을 지원하고 있지만 모델을 데이타베이스에 저장하여 공동으로 작업을 수행할 수 있게 하는 연구는 미흡하다. 따라서, 본 논문에서는 이 분야에 중점을 두고 소프트웨어 개발 주기와 연관하여 요구분석 단계의 산출물인 클래스, 사용 사례, 활동 다이어그램을 저장할 수 있는 관계형 데이타베이스를 설계하는 방법을 제시하고 이 방법을 이용한 개발 도구를 구현하였다.

3. UML 다이어그램 관리를 위한 데이타베이스 설계

UML의 표기법을 사용한 모델링 정보는 다수의 다이어그램들로 구성된다. 우선 전체 다이어그램들을 관리할 수 있는 다이어그램 관리 테이블이 필요하며 스키마 생성을 위한 규칙을 아래와 같이 정의한다.

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

[규칙 1]에서 집합 D는 다이어그램의 목록을 저장할 수 있는 집합으로써 각 다이어그램을 구별할 수 있는 다이어그램의 ID를 생성하여 식별자로 사용한다. 구현부분에서는 집합 D를 저장할 수 있는 테이블을 설계하고 이 식별자와 함께 다이어그램의 작성자, 작성일자와 같은 다이어그램에 대한 상세 정보도 저장되도록 한다.

각각의 다이어그램들은 다이어그램이라는 큰 틀 내부에 다수의 구성요소와 관계들이 존재하는 계층적인 구조로 분해할 수 있다. 관계형 데이타베이스에서는 이러한 계층적인 구조를 허용하지 않으므로 기본키와 외래키 정보를 이용하여 다이어그램을 저장할 수 있는 관계형 스키마를 설계해야 한다. 먼저 각 다이어그램의 구성요소들을 분류하는 규칙을 제시하고, 제시된 규칙에 따라 관계형 데이타베이스의 스키마를 생성한다. 따라서 전체 다이어그램을 관리할 수 있는 다이어그램 관리 테

이블과 각 다이어그램을 구성하고 있는 구성요소들을 분류하여 저장할 수 있는 테이블이 필요하며 이를 위해 스키마 생성을 위한 규칙들을 먼저 제시한다.

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 사용하여 인터페이스임을 명시하는 방법과 그림 2(b)와 같이 클래스의 스테레오타입(stereotype)에서 인터페이스를 명시하는 방법이 있다. 따라서 클래스 다이어그램을 관계형 스키마로 변환하기 위해서는 먼저 각 구성요소와 관계를 분류하는 규칙이 필요하며 본 논문에서는 다음과 같은 규칙들을 정의한다.

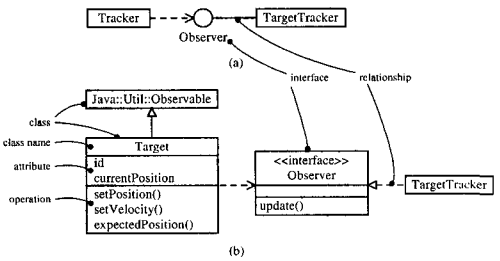

그림 2 UML 클래스 다이어그램

[규칙 2] 다이어그램 내의 클래스 집합은 $C=\{c_1, c_2, ..., c_l\}$로 정의하고, C의 임의의 원소인 클래스 $c_x=(class_id, is_interface, di)$으로 정의한다. 여기서, d_i는 c_x가 소속된 다이어그램이며 D의 원소이다(단, $x=1, 2, ..., l$).

[규칙 2]는 각 클래스 다이어그램 내부에 존재하는 클래스의 집합에 관한 것이다. 따라서 집합 C의 원소는 클래스의 식별자와 이 클래스들이 소속된 다이어그램을 나타낸다. 인터페이스는 다른 스테레오타입과 구분하기 위하여 인터페이스임을 구분하는 필드가 필요하며 항목 *is_interface*는 인터페이스의 두 가지 표기법을 구분할 때 사용한다.

[규칙 3] 클래스 내부의 속성 집합은 $A=\{a_1, a_2, ..., a_m\}$으로 정의하고, A의 임의의 원소인 속성 $a_x=(attribute_id, c_i)$으로 정의한다. 여기서, c_i는 a_x가 소속된 클래스이며 C의 원소이다(단, $x=1, 2, ..., m$).

[규칙 4] 클래스 내부의 연산 집합은 $O=\{o_1, o_2, ..., o_n\}$으로 정의하고, O의 임의의 원소인 연산 $o_x=(operation_id, c_i)$으로 정의한다. 여기서, c_i는 o_x가 소속된 클래스이며 C의 원소이다(단, $x=1, 2, ..., n$).

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

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

[규칙 5] 클래스 다이어그램 내부의 클래스간의 관계 집합은 $R_c=\{r_1, r_2, ..., r_o\}$로 정의하고, R_c의 임의의 원소인 관계 $r_x=(c_i, c_j, relationship, d_i)$으로 정의한다. 여기서, c_i는 상위 클래스(super class)이고 c_j는 하위 클래스(sub class)를 의미하고, d_i는 c_x가 소속된 다이어그램이며 D의 원소이다(단, $x=1, 2, ..., o$).

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

3.2 사용사례 다이어그램 변환

사용 사례 다이어그램은 행위자(actor), 시스템(system), 사용사례(use case)와 같은 구성요소와 각 구성요소간의 관계로 구성된다. 그림 3은 간단한 사용사례 다이어그램이다[17]. 사각형은 시스템을 나타내며, 타원은 하나의 사용사례이다. 사람 모양의 도형은 행위자를 나타낸다.

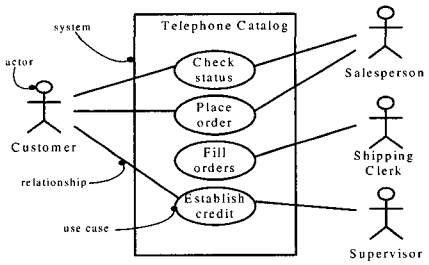

그림 3 사용사례 다이어그램

사용사례 다이어그램에서 사용되는 관계를 제외한 구성요소의 집합과 관계의 집합은 다음과 같이 정의할 수 있다.

[규칙 6] 사용사례 다이어그램의 구성요소인 행위자, 시스템, 사용사례의 집합은 $U=\{u_1, u_2, ..., up\}$

로 정의하고, U의 임의의 원소인 u_x=(*use_case_id, use_case_type, d_i*)으로 정의한다. 여기서, use_case_id는 관계를 제외한 구성요소를 나타내며 *use_case_type*에 의해서 구분된다. d_i는 u_x가 소속된 다이어그램이며 D의 원소이다(단, x=1, 2, ..., p).

[규칙 7] 사용사례 다이어그램의 구성요소간의 관계 집합은 R_u={r_1, r_2, ..., r_q}로 정의하고, R_u의 임의의 원소인 관계 r_x=(u_i, u_j, *relationship*, d_i)으로 정의한다. 여기서, ui는 관계의 시작 요소이고 u_j는 마침 요소를 의미하고, d_i는 r_x가 소속된 다이어그램이며 D의 원소이다(단, x=1, 2, ..., q).

[규칙 6]에서 *use_case_id*는 사용사례 다이어그램에서 행위자, 시스템, 사용사례의 식별자를 나타내며, *use_case_type*에 따라 행위자, 시스템, 사용사례로 구성요소들을 구별한다. [규칙 7]의 원소 u_i는 각 구성요소 사이에서 관계의 시작을 나타내며, u_j는 관계의 끝을 나타낸다. *relationship*은 두 원소 사이의 관계의 종류이다.

3.3 활동 다이어그램 변환

활동 다이어그램은 활동(activity), 구획면(swimlane), 액션(action), 객체(object) 등의 구성요소와 구성요소간의 관계로 구조를 표시할 수 있다. 그림 4는 소프트웨어 개발 프로세스를 활동 다이어그램으로 표현한 간단한 예제이다[17].

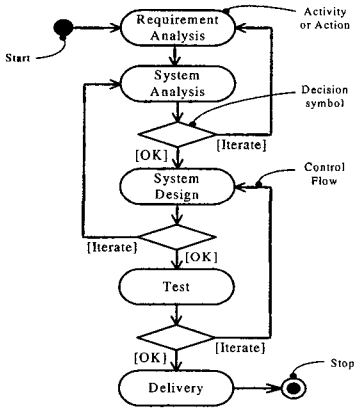

그림 4 활동 다이어그램

활동 다이어그램은 활동, 구획면, 액션, 객체 등과 같은 구성요소와 구성요소들간의 관계로 표현되며 각 구성요소들은 다양한 표기법을 가지고 있다. 따라서 [규칙 8]에서는 구성요소들의 집합을 정의하고, [규칙 9]에서는 구성요소들간의 관계를 정의하는 집합을 생성한다.

[규칙 8] 활동 다이어그램의 관계를 제외한 구성요소의 집합은 T={t_1, t_2, ..., t_r}로 정의하고, T의 임의의 원소인 t_x=(*activity_id, activity_type, d_i*)으로 정의한다. 여기서, *activity_id*는 활용 다이어그램에서 활동(activity), 구획면(swimlane), 액션(action), 객체(object) 등의 식별자를 나타내며, *activity_type*에 따라 각 구성요소들을 구분한다. d_i는 t_x가 소속된 다이어그램이며 D의 원소이다(단, x=1, 2, ..., r).

[규칙 9] 활동 다이어그램의 구성요소간의 관계 집합은 R_a={r_1, r_2, ..., r_s}로 정의하고, R_a의 임의의 원소인 관계 r_x=(a_i, a_j, *relationship*, d_i)으로 정의한다. 여기서, ai는 관계의 시작 요소이고 a_j는 마침 요소를 의미하고, d_i는 r_x가 소속된 다이어그램이며 D의 원소이다(단, x=1, 2, ..., s).

3.4 협력 다이어그램 변환

협력 다이어그램은 그래프에서 정점(vertex)처럼 상호작용에 참여하는 객체(object)와 객체들을 서로 연결하는 링크(link), 링크의 흐름 방향과 역할을 나타내는 메시지(message), 메시지의 순서(sequence)로 구조를 표현할 수 있다. 그림 5는 간단한 협력 다이어그램이다[10].

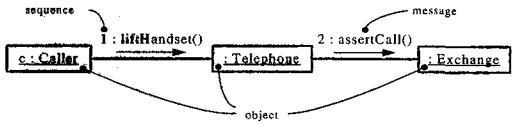

그림 5 협력 다이어그램

협력 다이어그램은 객체와 객체간의 상호 작용을 나타내는 링크, 메시지, 메시지 순서 등으로 표현된다. [규칙 10]에서는 객체들의 집합을 정의하고, [규칙 11]에서는 객체들간의 관계인 링크의 집합을 생성한다. [규칙 12]에서는 메시지의 순서와 함께 메시지의 집합을 생성한다.

[규칙 10] 협력 다이어그램의 객체들의 집합은 I={i_1, i_2, ..., i_t}로 정의하고, I의 임의의 원소인 i_x=(*object_id, object_type, d_i*)으로 정의한다. 여기서, *object_id*는 협력 다이어그램에서 객체(object)의 식별자를 나타내며, *object_type*에 따라 객체의 형태를 구분한다. d_i는 i_x가 소속된 다이어그램이며 D의 원소이다

(단, x=1, 2, ..., t).

[규칙 11] 협력 다이어그램의 객체들의 링크를 나타내는 집합은 L={l_1, l_2, ..., l_u}로 정의하고, L의 임의의 원소인 l_x=(*message_id*, i_i, i_j, d_i)으로 정의한다. 여기서, message_id는 메시지의 식별자이며, i_i는 링크의 시작 객체이고 i_j는 마침 객체를 의미하고, d_i는 l_x가 소속된 다이어그램이며 D의 원소이다(단, x=1, 2, ..., u).

[규칙 12] 협력 다이어그램의 메시지의 순서와 상호 작용을 나타내는 집합은 S={s_1, s_2, ..., s_v}로 정의하고, S의 임의의 원소인 s_x=(*sequence_id*, *sno*, l_i, *message*)으로 정의한다. 여기서, *message_id*는 메시지의 식별자이며, li는 연결된 링크를 의미한다(단, x=1, 2, ..., v).

3.5 관계형 데이타베이스 스키마

실제 구현을 위해서는 위에서 정의한 각 규칙에 필요한 정보를 추가하여야 한다. 예를 들면, 화면에 출력될 클래스는 화면상의 좌표 정보를 포함해야한다. 따라서 관계형 데이타베이스의 각 스키마(schema)를 생성할 때에는 추가적인 정보를 충분히 고려해야 한다.

그림 6은 다이어그램의 정보를 저장할 수 있는 관계형 스키마이다. DIAGRAM 테이블의 DID는 이후 생성될 클래스, 사용사례, 활동 다이어그램에서 참조하게 된다. DIAGRAM 테이블은 다이어그램을 작성한 사용자(OWNER) 정보를 이용하여 자신이 작성한 다이어그램을 검색할 수 있고, 공개 허용(ALLOW) 필드를 이용하여 다른 사람에게 작성한 다이어그램을 공개할 수도 있다. 또한 사용자의 추가 정보(INFO)를 사용함으로써 검색의 효율성을 높일 수 있도록 하였다.

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

그림 6 다이어그램 정보 저장 스키마

클래스 다이어그램 정보는 [규칙 2]부터 [규칙 6]까지의 정의에 추가적인 정보를 더하여 그림 7과 같은 예제 구문으로 생성한 테이블에 저장할 수 있다.

```
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 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
);

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 7 클래스 다이어그램 저장 스키마

CLASS 테이블의 DID는 DIAGRAM 테이블의 DID를 참조하는 외래키(foreign key)이다. 따라서 위의 두 키는 일대다의 관계를 지니게 된다. 또한 클래스의 화면 출력을 위해 클래스 표기의 시작 좌표(START_X, START_Y)와 UML의 모든 모델들은 스테레오타입(stereotype)을 가질 수 있으므로 저장할 수 있도록 고려하였다. 클래스 내부의 속성과 연산은 ATTRIBUTE와 OPERATION 테이블에 저장한다. 두 테이블에 저장되는 모든 값들은 CLASS 테이블의 기본키인 CID를 참조하여야한다. CLASS_RELATIONSHIP 테이블은 클래스나 인터페이스와 같은 모델 요소간의 관계를 저장하는 테이블로써, 다중성(multiplicity)과 역할명(role name)도 저장할 수 있도록 해야 한다.

그림 8은 사용사례 다이어그램의 정보를 저장할 수

있는 관계형 데이타베이스 스키마이다. USE_CASE 테이블에 있는 TYPE의 값에 따라 관계를 제외한 모든 구성요소들의 표기법을 구별한다. USE_RELATION SHIP 테이블은 사용사례 다이어그램 내부의 구성요소들 사이의 관계를 저장하는 테이블로서 SUPER_UID는 관계를 시작하는 구성요소를 나타내고, SUB_UID는 관계가 끝나는 구성요소를 나타낸다. TYPE은 관계의 종류를 구별하는 데 사용된다.

```
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 8 사용사례 다이어그램 저장 스키마

그림 9는 활동 다이어그램의 정보를 저장할 수 있는 관계형 데이타베이스 스키마로써 구성요소들은 ACTIVITY 테이블의 TYPE 값에 따라 구별되도록 하며, 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,
      START_X int,
      START_Y int
      INFO varchar,
);

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 9 활동 다이어그램 저장 스키마

그림 10은 협력 다이어그램의 정보를 저장할 수 있는 관계형 데이타베이스 스키마로써 객체들은 OBJECT 테이블에 저장하고, 링크는 LINK 테이블에 시작 객체와 마침 객체들을 저장하며, 각 메시지와 메시지 순서는 SEQUENCE 테이블에 저장한다.

```
create table OBJECT
(
      OID 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 LINK
(
      LID int primary key,
      SUPER_OID int foreign key references OBJECT(OID),
      SUB_OID int foreign key references OBJECT(OID),
      PATH_STEREO varchar,
      STEREOTYPE varchar,
      INFO varchar
);

create table SEQUENCE
(
      SID int primary key,
      LID int foreign key references LINK(LID),
      SNO varchar,
      MESSAGE varchar not null,
      INFO varchar
);
```

그림 10 협력 다이어그램 저장 스키마

그림 11은 위에서 설계한 스키마에서 참조 관계를 명시한 테이블의 구성도이다.

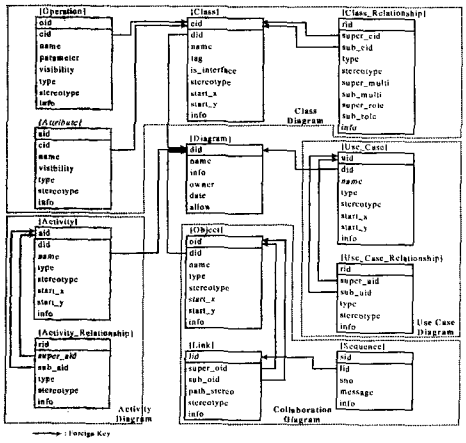

그림 11 테이블 구성도

4. 시스템 구현

본 논문에서 제시한 방법을 적용하여 구현한 시스템의 전반적인 흐름과 다이어그램의 검색방법으로 간단한 질의 조건을 처리하는 방법, 구현한 시스템을 사용하여 작성한 예제 다이어그램을 살펴보도록 한다.

4.1 시스템 흐름도

그림 12는 본 논문에서 제안한 방법을 적용하여 구현한 시스템의 흐름도이다. 클래스, 사용사례, 활동, 협력 다이어그램의 저장 및 검색은 동일한 흐름을 가진다. 사용자 인증 과정은 시스템의 보안을 위해서 필요하다. 다이어그램 모델을 작성한 후 저장 과정에서는 먼저 다이어그램의 정보를 입력받고, 이 후 작성한 다이어그램의 구성요소를 분류하여 각 테이블에 저장할 수 있도록 변환한다. 변환된 구성요소 정보는 각 테이블에 저장이 되며, 이 후 각 구성요소 사이의 관계 정보가 저장이 되도록 한다. 다이어그램을 검색하는 과정에서는 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.

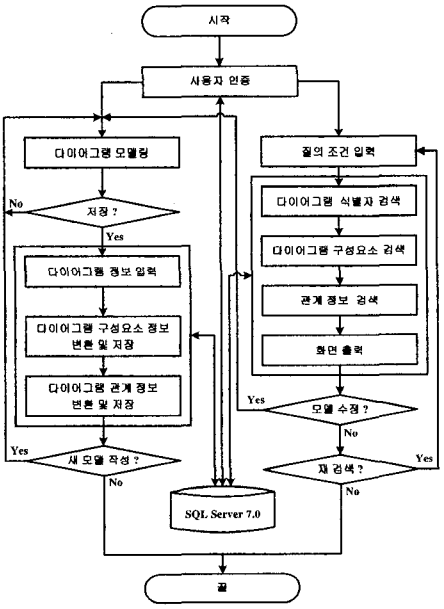

그림 12 다이어그램 저장 및 검색 흐름도

4.2 UML 다이어그램 검색

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

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, OPERATION 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 13은 클래스 다이어그램을 검색하기 위해서 클래스 이름이 조건으로 주어지는 경우 다이어그램을 검색하는 과정을 SQL 구문으로 간략하게 기술한 것이다.

조건 : 클래스의 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 13 클래스 다이어그램 검색 구문

4.3 시스템 구현

본 논문에서 제안한 방법을 구현하기 위하여 관계형 데이타베이스로는 Microsoft SQL Server 7.0을 사용하였고, 프로그래밍 언어로는 Visual C++ 6.0을 사용하였다.

그림 14는 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 데이타베이스와 연동하도록 구현하였다.

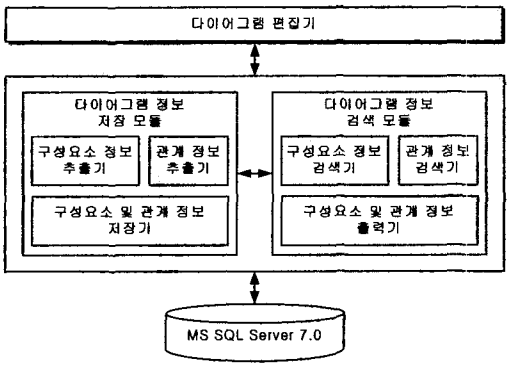

그림 14 다이어그램 관리 시스템 구조도

그림 15는 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.

제안한 시스템은 UML을 사용하여 설계된 모델링 정보를 다수의 사용자들이 동시에 공유하여 설계에 공동으로 참여할 수 있도록 하기 위해 기존의 파일 저장 시스템이 아닌 관계형 데이타베이스를 사용하도록 하였다.

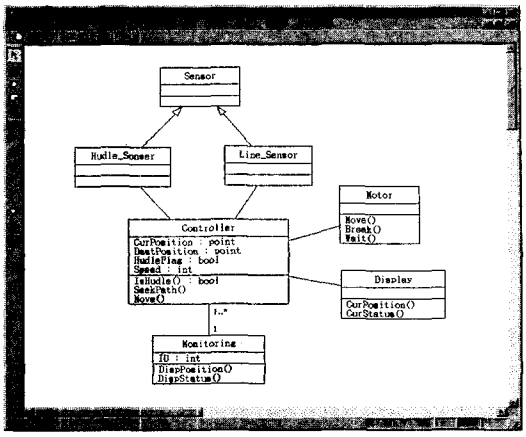

그림 15 구현한 시스템을 사용한 예제 클래스 다이어그램

기존의 파일 저장 시스템을 사용하고 있는 Rose나 Plastic은 모델링 정보를 동시에 공유할 수 없다는 단점이 있다. 현재는 지원하는 다이어그램의 수는 4개이지만 본 논문에서 제안한 방법을 적용하면 나머지 다이어그램들도 데이타베이스를 이용하여 저장하고 관리하는 데 어려움은 없을 것이다.

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 데이타베이스를 사용하도록 하였다. 기존의 UML 지원도구인 Rose나 Plastic은 UML에서 지원하는 모든 종류의 다이어그램을 지원하고 있으며, 프로그램 코드로부터 다이어그램을 생성하는 순공학과 다이어그램으로부터 프로그램 코드를 생성하는 역공학을 지원하고 있다. 하지만 이 두 시스템은 파일 저장 시스템을 사용함으로 인하여 동시 공용 즉, 실시간에 모델링 정보를 공유할 수 없다는 단점과 함께 기존에 작성된 모델링 정보를 다양한 조건의 질의어를 사용하여 검색하는 데 어려움이 있다.

제안한 시스템은 데이타베이스를 지원함으로써 다수의 개발자들이 실시간으로 접근 가능하도록 하였으며, 이미 작성된 모델의 부분 정보만으로도 검색이 가능하도록 함으로써 모델 재사용성을 높일 수 있다. 또한 데이타베이스에서 지원하는 트랜잭션을 사용함으로써 장애가 발생하였을 경우에 이전 정보를 복구할 수 있다는 장점과 모델의 버전관리에도 편리성을 제공할 수 있다. 표 1은 Rose, Plastic 그리고 제안한 시스템을 제공하는 기능면에서 비교하고 있다.

표 1 Rose, Plastic과 제안된 시스템과의 비교

시스템 / 지원 분야	Rational Rose 2000 Enterprise	Plastic 3.0	제안 시스템
지원 다이어그램 수	9	9	4
순공학 지원	○	○	✗
역공학 지원	○	○	✗
데이타베이스 지원	✗	✗	○
실시간 정보 공유	✗	✗	○
트랜잭션 지원	✗	✗	○

5. 결론 및 추후 연구과제

UML은 기존의 다양한 시스템 개발 방법론들을 하나의 틀로 통합한 것이다. 특히, 소프트웨어 시스템의 요구분석부터 설계 및 개발 등을 체계적으로 지원하는 모델링 언어이다. 이러한 UML로 개발된 모델들의 효율적인 사용을 위해서는 모델들을 통합하여 저장하고 관리할 필요성이 있다.

본 논문에서는 소프트웨어 생명주기의 요구분석 단계에서 사용되는 UML 다이어그램을 저장하여 관리할 수 있는 관계형 데이타베이스를 설계하였고, 검색하는 방법을 제안하였다. 먼저 클래스, 사용사례, 활동, 협력 다이어그램을 구성하고 있는 각각의 요소들을 분류하여 테이블로 구성하고 관계형 데이타베이스의 기본키와 외래키 정보를 이용하여 각 테이블의 관계를 유지하는 방법을 사용하였다. 제안한 방법에 따라 모델링 정보를 데이타베이스화함으로써 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 데이타베이스를 사용함으로써 현재의 데이타베이스 시스템에 바로 적용할 수 있다는 장점도 가지게 된다.

추후 연구과제로는 현재 요구분석 단계에서 적용한 방법들을 확장하여 설계 및 구현 부분까지 데이타베이스와 연동할 수 있도록 확장하는 것이다. 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) 데이타베이스에서도 UML 다이어그램 정보를 저장 및 검색할 수 있도록 해야 할 것이다.

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, vol. 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] Rational Software's UML Resource Page, http://www.rational.com/uml

[6] Plastic Software's Home Page, http://www.plasticsoftware.com

[7] Z. Xie, J. Yu and J. Liu, "Applying UML to Gas Turbine Engine Simulation," Proc. of TOOLS 31, pp.458-464, Sep. 1999.

[8] R. Jigorea, S. Manolache, P. Eles and Z. Peng, "Modelling of Real-Time Embedded Systems in and Object-Oriented Design Environment with UML," Proc. of the 3rd IEEE Int. Symposium on Object-Oriented Real-Time Distributed Computing, pp.210-213, Mar. 2000.

[9] OMG UML Specification v.1.4 draft, http://www.omg.org

[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] 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] B. Demuth and H. Hussmann, "Using UML/OCL Constraints for Relational Database Design," UML'99: The Unified Modeling Language-Beyond the Standard, pp.598-613, Oct. 1999.

[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, pp.91-108, Oct. 1999.

[17] H. Eriksson and M.Penker, Business Modeling with UML, OMG Press, 2000.

이 성 대

1999년 한국해양대학교 컴퓨터공학과(공학사). 2001년 한국해양대학교 컴퓨터공학과(공학석사). 2001년 ~ 현재 한국해양대학교 컴퓨터공학과 박사과정. 1995년 ~ 1996년 미래정보CIM. 관심분야는 데이타베이스, 데이타마이닝, UML

박 휴 찬

1985년 서울대학교 전자공학과(공학사). 1987년 한국과학기술원 전기및전자공학과(공학석사). 1995년 한국과학기술원 전기및전자공학과(공학박사). 1987년 ~ 1990년 금성반도체 연구원. 1997년 ~ 현재 한국해양대학교 기계·정보공학부 조교수. 관심분야는 데이타베이스, 모델링방법론, UML