통합 관리 모델을 이용한 효율적인 객체 관리 저장소 설계

선 수 균[†] · 송 영 재[††]

요 약

최근 컴퓨팅 환경은 통합되는 개방형 시스템으로 변모하고 있다. 기존 시스템에서는 다양한 개발 환경을 지원하지 못하고 있으며, 다양한 산출물(클래스 부품, 다이어그램, 양식, 컴포넌트)에 대한 효율적인 관리를 못하는 단점이 있다. 또한 산출물에 있어서 연결관계를 서로 관련성 있는 사항을 참조 못하고 있으며 객체 내부의 관계를 정의 할 수 없어서 산출물에 대한 이해도가 부족으로 객체지향 프로그램에 필요한 코드로 생성하지 못하고 있다. 따라서 본 논문에서는 산출물 객체의 효율적인 관리를 위한 통합 관리 모델을 이용한 객체 관리 저장소 설계기법을 제시하고, 버전 관리와 연결관계 상호관련기를 두어 객체내부 관계와 서로 관련성을 개발자가 쉽게 파악하여 재사용함으로서 개발자는 어플리케이션에 관련된 부분만을 개발할 수 있도록 지원한다. 객체 관리 저장소를 네 가지 단계로 설계함으로서 단점을 보완하고 클래스이해와 산출물 정보를 쉽게 표현할 수 있어 클래스를 생성할 수 있도록 하여 재사용의 효율성을 향상 시켰다. 또한 새로운 소프트웨어 개발에 생산성을 향상시키기 위한 것이 본 연구의 목적이다. 향후 시스템의 통합에 잇점인 소프트웨어의 재사용성을 극대화하여 생산성을 향상시키는 프로토타이핑을 지원할 것으로 기대된다.

A Design of Efficient Object Management Repository Using Integration Management Model

Su-Kyun Sun[†] · Yong-Jea Song[††]

ABSTRACT

Lately computing environment is changing into integrating open system. This paper proposes Integrated Management Model to improve productivity about new software development. The model is divided by Management Model to deal with the rapidly changing environment effectively into three layers: the first layer classifies and displays information to users, the second layer controls function, the integration and management layer, and the last layer manages data, the object management storage layer. So it designs of Efficient Object Management Repository Using Integration Management Model. This might support afterward prototyping in maximizing the reuse of software, which is advantage to the integration of the system, and in promoting its productivity.

키워드 : 통합 관리 모델(Integrated Management Model), 객체 관리 저장소(Object Management Repository), 버전(Version), UML(Unified MOdeling Lanague), 컴포넌트(Component), 관계정보(Relation_information)

1. 서 론

최근에 객체 지향 기법이 확산되고 있다. 1980년대 중반부터 소프트웨어 위기의 극복을 위해 대두된 객체 지향 패러다임(object-oriented paradigm)의 영향에 의해 객체 지향 프로그래밍 언어와 객체 지향 소프트웨어 공학 기술이 급속도로 발전하고 있다. 객체 지향 소프트웨어의 개발을 위한 객체 지향 소프트웨어 공학 방법론으로는 1995년 이후로 Rumbaugh의 OMT(Object Modeling Technique), Jacobson의 Objectory, Booch의 OOA/OOD, 이들의 방법을 취합한 UML(Unified Modeling Lanague)등이 다양하게 제시되었다[1-3, 7, 8].

객체 지향 소프트웨어 개발하는데 필요한 많은 객체들이 요구되고 있으며, 그 과정 중에서 문서, 다이어그램, 원시코드, 방법론 정보, 설계 패턴등의 프레임워크, 사용자 인터페이스 객체, 데이터베이스 스키마 등의 다양한 산출물이 생성되고, 이러한 객체들을 효율적으로 저장, 관리, 검색, 재사용하는 모델이 필요하게 되었다. 과거 대부분의 제품은 Microsoft 플랫폼에 종속적이며, 현재 Visual Basic 개발 환경을 위한 컴포넌트(component)의 지원과 탐색을 위주로 지원하고 있어 C++이나 Java와 같은 다양한 개발 환경을 지원하지 못하고 있고, 다양한 산출물을 모두 관리하지 못하는 단점이 있다.

† 종신회원 : 동원대학 사무자동화과 교수
†† 종신회원 : 경희대학교 전자계산공학과 교수
논문접수 : 2001년 1월 19일, 심사완료 : 2001년 4월 4일

"재사용을 위한 처리기"[9] 역시 텍스트적인 방법론으로 구조적 방법론을 사용한 Teamwork[3]과 비교하고 있다. 클래스 부품의 추출과정에 있어서는 클래스들의 계층관계를 나타내는 추출 Viewer 기능이 정보의 표현에 한정되어 있어서 부품들의 생성, 삭제, 삽입 기능이 없어서 그 효율성이 떨어지고 있다. 또한 클래스 추출에 있어서 관련성 있는 사항을 함께 추출하지 못하는 단점이 있고, 객체 모델을 객체지향 프로그램에 필요한 코드로 생성하지 못하고 있다. 따라서 클래스 부품을 보다 효율적으로 저장하기 위한 객체를 추출하는 동시에 연결관계를 형성할 수 있는 기능을 추가하며 객체와 객체간, 객체 내부간의 모델링과 저장, 관리, 검색하는 과정에서 개발자가 필요한 사항을 빠르게 재사용할 수 있는 효율적인 객체 관리 저장소가 필요하게 되었고 이를 지원하는 템플리트 라이브러리 관리가 필요하게 되었다[4-6].

따라서 본 논문에서는 이러한 문제점을 개선하기 위하여 클래스 부품의 분석 정보를 규칙적이고 효율적으로 저장하고 Viewer 기능을 이용해 재사용성을 향상시킬 수 있도록 클래스 부품의 다양한 정보를 제공하는 동시에 연결 관계에서 구현하기 어려운 양방향 관련성도 쉽게 정의 할 수 있도록 버전단계와 "연결관계 상호 관련기"를 두었다. 기존의 정보 저장소의 단점을 보완하고 객체 관리 저장소를 설계하여 데이터 단계, 프로세스 단계, 뷰어 단계, 버전 단계로 세분화하였다. 버전단계는 객체 지향 프로그램에 필요한 코드로 생성하기 위해서 객체모델의 모델링을 객체와 객체간 및 내부간의 관계를 형성 관리하는 단계를 형성한다.

본 논문의 목적은 다음과 같다. 첫 번째는 통합 관리 모델을 이용해서 객체 지향 소프트웨어 공학 프로세스에서 발생하는 산출물을 데이터 베이스화한 효율적인 객체 관리 저장소 설계하는 기법제시이다. 두 번째는 개발에 필요한 산출물들을 저장할 수 있는 모델제시이다. 세 번째는 기존의 단점인 객체와 객체간의 구조와 객체간의 내부 모델간의 관련된 정보가 없어서 효율적인 관리를 할 수 없었던 것을 객체 내부 메타 모델링하여 개발하는 필요한 산출물들을 생성, 관리, 검색, 저장할 수 있는 객체저장소 모델을 제안하고, 설계하는 기법을 제시한다.

마지막으로는 객체 지향 소프트웨어 공학 프로세스에서 발생하는 다양한 산출물을 데이터베이스화하여 필요한 산출물을 용이하게 검색하게 하고, 다양한 CASE 도구와 사용자들이 이러한 산출물들을 동시에 공유하게 함으로써, 궁극적으로 객체 지향 소프트웨어 개발의 생산성을 대폭 개선시키는 데 있다.

2. 관련 연구

2.1 모델링을 이용한 연결관계 관련성 관리자

"모델링 패턴을 이용한 연결관계 관련성 관리자"[10]에서는 객체모델링의 가장 기본적인 관련성인 연결관계를 기반으로 객체지향 프로그래밍에 필요한 코드를 생성하는 템플리트에 대하여 정의하였다. 이러한 템플리트를 활용하여 클래스를 구현하면 연결 관계와 클래스의 본질간을 분리할 수 있으며, 양방향 관련성을 정의 할 수 있고, 연결관계 정의의 일관성을 보장 할 수 있다. 그러나 프로그래머가 연결관계의 템플리트를 이해하고 모델에 필요한 연결관계를 직접 템플리트를 활용하여 구현해야 한다면 이는 패턴을 이해하고 이를 자신의 문제에 활용하는 만큼의 노력이 요구될 수도 있다. 프로그래머가 템플리트를 활용하는 방법을 습득하기 위한 노력을 줄여 주고 어플리케이션에 관련된 부분만을 개발할 수 있도록 지원하고, 변경이 발생하는 연결관계에 따라 템플리트에 의해 개발된 코드의 변경을 책임지는 연결 관계 관련성 도구가 연결 관계 관련성 관리자다. 연결관계 관련성 관리자[10]는 두 가지 서브 시스템으로 나뉜다. 서브 시스템 관련성 모델 와 서브 시스템 통신 모델이다.

서브 시스템 관련성 모델(Subsystem Relationship Model)은 서브시스템에 대한 전체 객체 관련성 다이아 그램을 표현하는 것이고, 서브 시스템 통신 모델(Subsystem Communication Model)은 서브 시스템간의 메시지를 주고받는 관계를 표현하고 있다.

이들 세 모델 역시 분석 단계의 객체 단위의 정보 모델, 통신 모델을 서브 시스템 단위의 모델로 확장한 것이다. 이에 따른 시스템 설계의 산물로 서브 시스템간의 관련성 모델로 표시한 것이고, 상위 수준의 메시지 송수신 관계를 설명한 것이 서브 시스템간의 통신 모델이다.

2.2 정보 저장소

정보저장소[11]는 저장되는 각각의 클래스 정보를 데이터(Data), 프로세스(Process), 뷰어(Viewer)의 3단계로 분류하여 구성하였다. 데이터는 구문분석으로 추출된 객체의 클래스 단위로 이루어진다. 프로세스 수행 과정은 소스 코드를 읽어 각 클래스를 추출하여 클래스 사이의 관계를 계층적으로 표현해 준다. 이때 각 클래스는 자신이 가지고 있는 부품 정보(클래스명, 멤버함수, 속성, 원시코드)와 함께 추출된다. 사용자는 클래스 계층도에서 자신이 원하는 클래스를 삭제하고 추가하는 기능을 이용하여 새로운 계층도를 만들 수 있다. 정보저장소는 크게 객체의 정보와 다이어그램 정보로 구성된다. 객체의 정보는 클래스, 멤버함수(method), 데이터 형(data_type)으로 구성되어 구문분석 단계에서 DB 파일에 저장된다. 다이어그램 정보는 기호(symbol), 사건(event), 관계정보(relation_information), 위치(location), 클래스의 수(class_number) 등으로 구성된다. 다이어그램 정보를 이용하여 그래픽으로 표현할 때 나타나는 객체들은 객체 내부의 정보뿐만 아니라 객체들 사이의 상

속관계 정보와 이의 표현을 위한 위치정보도 포함하고 있어야한다. 이들 정보를 바탕으로 새롭게 생성된 객체들을 다시 계층도로 표현해야 하기 때문이다. 이처럼 정보저장소는 구문분석을 통한 클래스 정보와 View를 위한 정보로 구성된다.

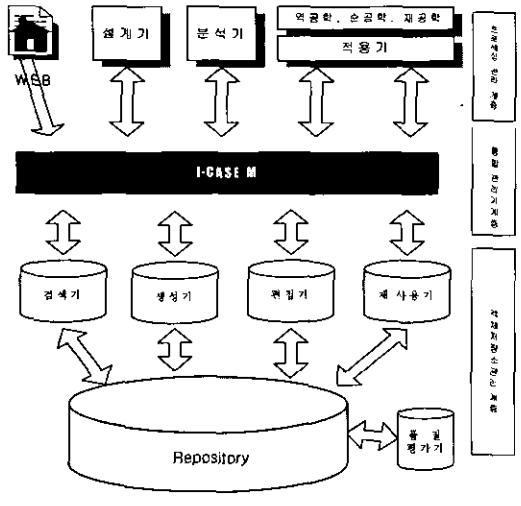
2.3 UML(Unified Modeling Language)

UML은 소프트웨어 시스템이나 Business Modeling 그리고 기타의 비 소프트웨어 시스템 등을 나타내는 산출물들을 구체화하고, 시각화하고, 구축하고, 문서화하기 위해 만들어진 언어로서, 복잡하고 거대한 시스템을 모델링함에 있어 성공적으로 증명된 공학적인 경험들을 포함하고 있다. UML은 OMT, Booch, OOSE/Jacobson에서 발견되는 모델링 언어의 장점들을 계승하여 만든 언어이고, 객체 기술에 관한 국제 표준화 기구인 OMG(Object Management Group)에서 표준화로 인정하고 있으며 따라서 산업표준으로 정착하고 있는 실정이다[2, 7].

3. 통합 관리 모델

통합 객체 관리 모델[12]을 더 확장한 것이 통합 관리 모델(Integrated Management Model)이다.

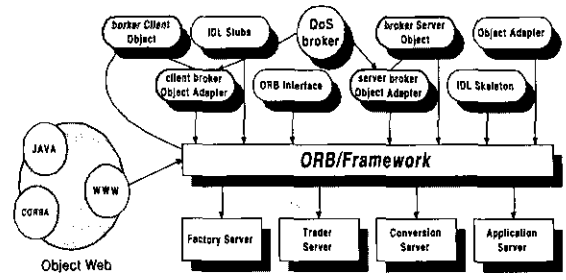
(그림 1)은 통합 관리 모델 전체 구성을 나타낸 것으로 세 가지 계층으로 구성되는데, 위 계층을 프로세싱 관리 계층, 가운데 계층을 통합 관리기 계층(I-CASE Manager), 아래 계층을 객체 저장소 관리 계층이다.



(그림 1) 통합 관리 모델 전체 구성도

첫 번째는 프로세싱 관리 계층이다. 산출물의 분류, 설계, 분석, 적용에 필요한 곳을 다룬다.

두 번째는 통합 관리기 계층이다. 기존 시스템과 통합 및 표준화 작업, 산출물들을 제어하는 제어기능으로 분산 객체 환경에 적합한 구조로 이기종간 쉽게 통합할 수 있다.



(그림 2) 통합 관리기(I-CASE M) 구성도

세 번째는 객체 저장소 관리 계층이다. 이것은 급변하는 소프트웨어 환경에 대처하고 객체관리를 효율적으로 관리하며 개발에 필요한 산출물들을 저장할 수 있는 객체저장소 관리 계층이다. 즉, 데이터를 관리하고 객체 지향 소프트웨어 공학 프로세스에 의해 생성되는 제반 산출물을 객체 형태로 통합 관리하는 데이터베이스이다.

본 연구에서는 통합 관리 모델 중 세 번째 계층인 객체 관리 저장소를 더욱 보강하고 세분화하여 개발자가 다양한 산출물을 재사용 할 수 있도록 추출단계에서 재구성 단계로 가고 다시 저장단계에서 검색단계로 가는 순환 기능을 더욱 강화하여 개발자가 필요한 사항을 쉽게 추출,검색, 관리, 저장하여 소프트웨어 개발의 생산성을 더욱 높일 수 있도록 시스템을 제시한다. 객체 관리 저장소는 저장되는 각각의 클래스 정보를 [11]에서 제안한 데이터(Data)단계, 프로세스(Process)단계, 뷰어(Viewer)단계를 기본으로 하고, 메타모델링 과정을 반영한 버전(Version)단계을 더 추가시킨 것이다.

객체 관리 저장소의 설계를 제시한 목적은 다음과 같다.

첫째는 [11]의 단점을 보완하기 위하여 클래스 부품의 정보화를 다양화하고 규칙화하는 설계 기법을 제시한다. 각 클래스와의 상속관계를 중심으로 정보를 추출하고, 동시에 자체정보, 관계정보, 다이어그램 정보를 클래스 원시코드에 적용시켜 새로운 클래스 생성, 기존의 클래스 삭제, 삽입이 가능하도록 했다. 둘째는 각 클래스와의 관계를 쉽게 알 수 있도록 "연결 관계 상호 관련기"로 하여금 관계 정보를 제시하고 이것은 응용 모델에서 연결관계는 지속적으로 변화가 일어날 수 있어 개발자가 신속한 연결 관계를 알 수 있도록 객체 정보를 상세화한 것이다.

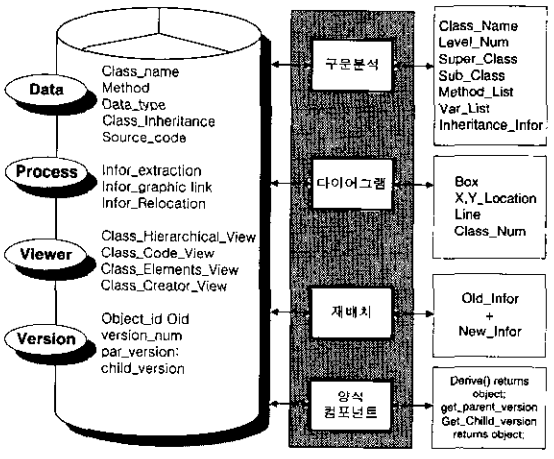
4. 객체 관리 저장소 설계

본 연구에서는 [12]에서 제안한 통합 객체 관리 모델을 이용해서 객체 관리 저장소를 설계하는데 효율적인 객체 관리를 위해서 객체에 대한 메타 모델링, 객체간의 관계에 대한 메타 모델링, 객체내부에 대한 메타 모델링이 진행되어야 한다. 이것이 버전단계이다. 객체 관리 저장소는 정보저장소[11]를 더 확장시킨 것으로 객체 관리 저장소 모델을 제안하고, 설계 기법을 제시한다. 객체 관리 저장소의 전체

구성은 (그림 3)과 같다. 이것은 저장되는 각각의 클래스 정보를 데이터(Data)단계, 프로세스(Process)단계, 뷰어(Viewer)단계, 버전(Version)단계의 4단계로 구성돼 있다.

4.1 데이터 단계 설계

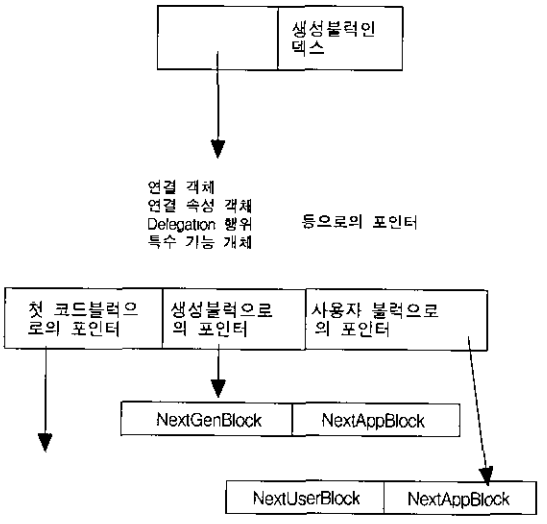
데이터 단계는 구문분석으로 추출된 객체의 클래스 단위로 이루어진다. 이들 정보를 바탕으로 새롭게 생성된 객체들을 다시 계층도로 표현해야 하기 때문이다. 이처럼 정보 저구문분석을 통한 클래스 정보와 View를 위한 정보를 제공하는 단계이다. 이 단계에서는 클래스 구문 분석/ 정보 추출/ 클래스 명 추출[11]의 방법을 적용했다. 다른 방법으로는 컴파일러나 프로그램 도구 내에 이미 내재된 것도 많이 볼 수 있다. 그 속성 값만 읽어 내면 쉽게 구할 수 있는 부분이기 때문에 본 논문에서는 생략하기로 한다.



(그림 3) 객체 저장소 모델

4.2 프로세서 및 View 단계 설계

프로세스 단계는 소스 코드를 읽어 각 클래스를 추출하여 클래스 사이의 관계(relationship)를 계층적으로 표현해 주며, 클래스의 구문분석과 View를 위한 정보를 처리하는 과정이다. View 단계는 각 클래스를 자신이 가지고 있는 부품 정보(클래스 이름, 클래스 식별자, 속성 리스트, 슈퍼 클래스와 서부 클래스, 상속관계, 행위 배열의 포인터등)와 함께 추출하는 과정도 포함하고 있다. 또한 자신이 원하는 클래스를 삭제하고 추가하는 기능을 이용하여 새로운 계층도를 만들어 주는 기능을 하는 단계이다. 그러나 [11]에서는 연결관계의 관련사항이 매우 미약해서 클래스 정보와 클래스 사이의 관계정보, 기호정보, 위치정보를 통합 할 수가 없었고 클래스자체를 재사용 할 수 없었다. 또한 연결관계에서 구현하기 어려운 양방향 관련성도 쉽게 정의 할 수 없었다. 따라서 본 논문에서는 연결관련사항을 관리할 수 있는"연결 관계 상호 관련기"를 정의를 해서 템플리트 라이브러리를 이용하여 상속관계와 클래스의 관계, 객체간의 연결관계를 세분화하고 이런 단점들을 해결한 것이 "연결 관계 상호 관련기"이다. 이것을 이용하면 개발자의 부담을 줄일 수 있었고 클래스 정보와 클래스 사이의 관계정보, 기호정보, 위치정보를 통합해서 연결관계와 클래스의 본질간을 분리할 수 있기 때문에 클래스 자체를 재사용하기가 매우 용이하다. 지속적으로 변화가 일어날 때 연결관계 부분에 대한 정의도 쉽게 변형할 수 있다. 또한 상속 및 관계의 상세화로 인한 재사용성을 극대화시킨 것이 장점이다. "연결 관계 상호 관련기"는 각 연결관계를 추출하여 이에 관련된 연결관계 인스탄스를 생성하며 또한 이에 해당하는 코드를 생성하여 어플리케이션에 등록하는 단계이다.



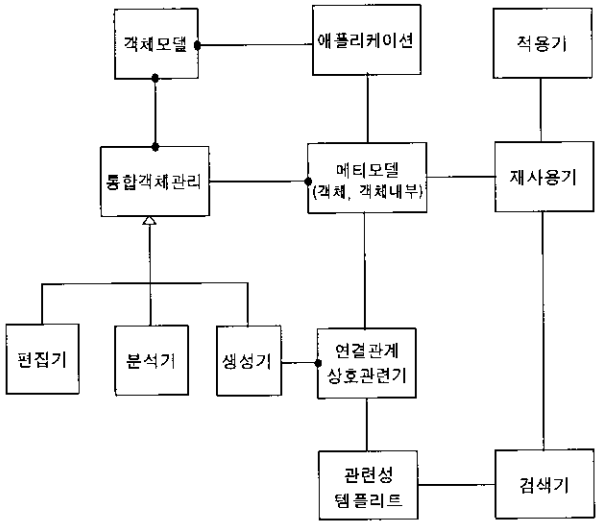
(그림 4) 연결 관계 상호관련기의 리스트 구조

(그림 4)는 연결 관계 상호 관련기를 생성블럭과 사용자 블록으로 나누어진 리스트 구조를 나타낸 것으로 사용자 블록은 사용자가 직접 정의한 코드를 의미하며, 포인터를 이용하여 서로 연결할 수 있다. 또한 생성블럭은 변형기에 의해 템플리트를 이용하여 생성한 코드를 의미하며, 서로 연결할 수 있도록 포인터를 이용한다.

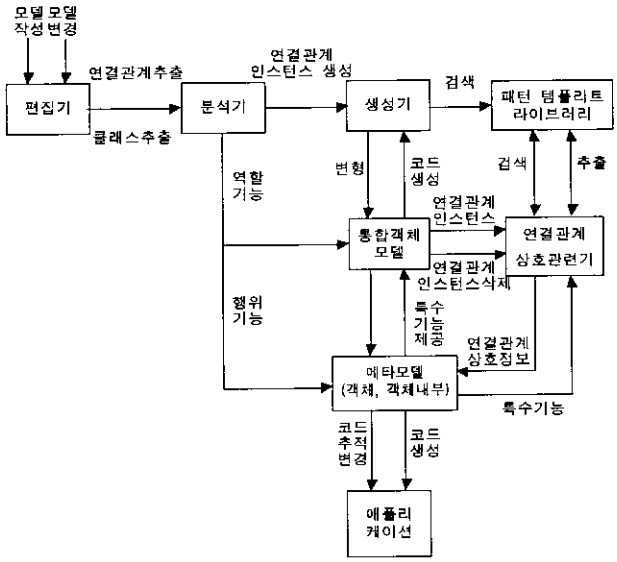
"연결 관계 상호 관련기"는 [10]에서 언급한 *연결관계 관련성 관리자*를 더 확장시킨 모델로 두 가지 서브 시스템으로 나뉜다. 하나는 서브 시스템 관련성 모델(그림 5)이고 나머지 하나는 서브 시스템 통신 모델(그림 6)이다. (그림 5)은 서브시스템에 대한 전체적인 것을 객체 관점에서 서로 관련성이 있는 것을 다이어그램으로 표현하는 것이고, (그림 6)은 서브 시스템간의 메시지를 주고받는 관계를 통신 관점에서 표현한 것으로, 상위 수준의 메시지 송수신 관계를 설명한 것이다.

(그림 5) 및 (그림 6)에서와 같이 객체모델은 관리기의 일종인 편집기에 의해 작성되어 지고, 분석기에게 연결관계 추출을 요구하면 분석기는 모델에서 연결관계를 추출하여 이를 생성기를 통해 연결관계 인스탄스를 생성하도록 한다. 생성기는 통합 객체 모델을 통하여 객체와 객체 내부의 관련사항을 파악하여 양방향의 관계를 파악한다. 또한 통합

객체 모델에서 연결 관계 인스탄스를 생성하여 연결 관계 상호 관련기를 통하여 서로 관련된 객체를 패턴 템플리트 라이브러리에 저장하여 추출, 검색한다. 또한 이에 해당하는 코드를 생성하여 어플리케이션에 등록한다. 후에 다시 사용자가 편집기를 통하여 기존의 모델을 변경하면 분석기는 변경 사항을 파악하여 메타모델을 통해 어플리케이션에 등록된 코드를 추적하여 해당되는 코드를 변경하고 연결관계 상호 관리기에 등록한 인스탄스를 관리하는 것이다. 따라서 "연결 관계 상호 관련기"는 다음과 같은 장점이 있다. 첫째, 연결관계와 클래스의 본질간, 또는 객체간을 분류할 수 있기 때문에 라이브러리에 저장된 템플리트를 활용하여 초기 프로그램 코드를 생성하도록 지원한다. 둘째, 수퍼 클래스와 서브 클래스간을 쉽게 알 수 있기 때문에 연결관계 변화를 개발자가 직접 수정하는 노력을 줄여준다.



(그림 5) 서브 시스템간의 관련성 모델



(그림 6) 서브 시스템간의 통신 모델

4.2.1 편집기 설계

편집기는 "연결 관계 상호 관련기"를 작성 및 수정할 수 있도록 지원하는 서브시스템으로 추후 생성한 템플리트 기반 코드와 연계성을 관리하기 위한 구조가 필요하고, 객체 관리 저장소에 저장시키기 위한 권리 구조로 다음과 같은 구조를 가져야 한다.

(a) 각 클래스에 대한 저장 구조

클래스 이름	클래스 식별자	속성 리스트	슈퍼 클래스	서브 클래스	상속 관계	행위 배열로의 포인터	역할 객체 리스트로의 포인터

(b) 역할 객체에 대한 저장 구조

역할 객체 식별자	ThisClass로의 포인터	Participant Class로의 포인터	Associative Class로의 포인터	다중성	선택적	Delegation 행위 플래스로의 포인터

(c) 행위 배열

클래스 이름	인덱스	함수 포인터

(d) Delegation 행위 클래스

이름	ThisClass로의 포인터	Participant Class로의 포인터	ToDelegation 행위로의 포인터	FromDelegation 행위로의 포인터

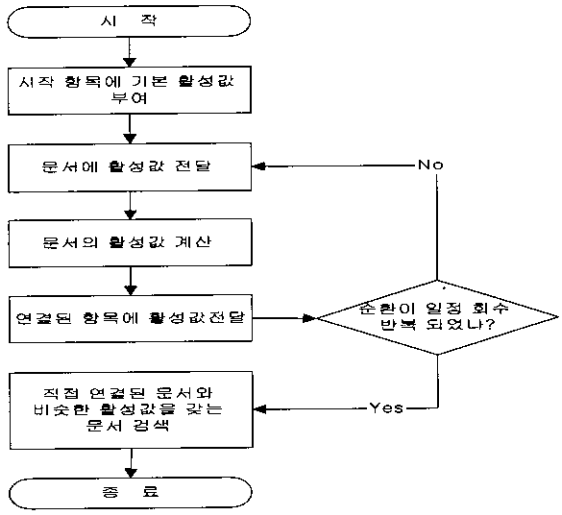
(e) 산출물 객체에 대한 저장 구조

클래스 이름	산출물 식별자	활성값	설명	행위배열로의 포인터	역할 객체리스트로의 포인터	작성자	작성시간 날짜

(그림 7) 객체 저장소에 저장시키기 위한 관리 구조

편집기에 의해 작성된 모델은 분석기에 의해 분석되고, 저장되며 이렇게 저장된 형태는 다시 생성기에 의해 수정이 필요할 때 편집기에 의해 모델로 재구성된다.

4.2.2 검색기 설계



(그림 8) SARM

본 연구에서 검색기 설계 방법은 어휘문제를 피하면서 효과적인 검색 결과를 얻기 위해 SARM(Spreading Activation Retrieval Method)[11]을 이용한다.

SARM에서는 항목과 연관 있는 문서들을 찾아내기 위해 활성값을 이용한다. 활성값은 각 항목과 문서들에 대해 계산되며 비슷한 값을 갖는 항목과 문서는 관련 있는 것으로 본다.

본 논문에서는 산출물 구조에 이 활성값을 저장함으로서 객체 관리 저장소의 검색시 적용한다.

SARM은 비연속적인 시간 단위에 대응하는 사이클에 대한 문서와 항목의 활성 레벨에 의해 정의된다. 시간 $t+1$일 때의 문서 D_i의 활성값은 다음과 같이 주어진다.

$$D_i(t+1) = \begin{cases} \delta_{D_i} t + \Psi_{D_i}(t)(M - D_i(t)) & if\ \Psi_{D_i}(t) < 0 \\ \delta_{D_i} t + \Psi_{D_i}(t)(D_i(t) - m) & if\ \Psi_{D_i}(t) \le 0 \end{cases}$$

여기서 $\delta_{Di}(t)$는 decay rate에 의해서 감소된, 시간이 t일 때의 활성값이다. 이것은 아래와 같다.

$$\delta_{D_i} = (1 - \Theta_D) D_i(t)$$

그리고, $\Psi_{Di}(t)$는 시간 t에서 문서 i에게 들어오는 입력으로 다음과 같이 주어진다.

$$\Psi_{D_i}(t) = \left(\frac{\overline{C}_D}{C_{Di}} \right)^{a_D} \sum_{j=1}^{n_T} W_{ji} U(T_j(t))$$

위 식에서 $U(x)$는 음수의 활성값을 걸러내기 위한 함수로서 이 함수에 의해 음수의 활성값은 영향을 미치지 않게 된다.

$$U(x) = \begin{cases} 0 & \text{if } x \ge 0 \\ x & if\ x > 0 \end{cases}$$

결과값은 -0.2와 1의 사이에서 활성값이 유지되는데 이 비대칭의 값은 양수 활성값의 흐름을 원활하게 하는데 요구된다.

활성값을 계산하는데 사용되는 많은 파라미터들은 <표 1>에 정의되어 있다.

〈표 1〉 SARM과 관련 있는 파라미터

기 호	정 의
n_D	정보저장소의 전체 문서의 수
n_T	정보저장소의 전체 항목의 수
w_{ij}	항목 i와 문서 j의 연결강도
Θ_D	문서 decay rate = 0.1
Θ_T	항목 decay rate = 0.25
M	최대 활성값 = 1.0
m	최소 활성값 = -0.2
C_{Di}	문서 i 의 항목의 수
C_{Tj}	항목 j의 문서 수
a_D	문서 fan-in exponent = 0.1
a_T	항목 fan-in exponent = 0.3
df_j	항목 j의 문서 빈도

4.3 버전 관리

본 논문에서는 통합 관리 모델 중에서 객체 관리 저장소를 설계하는 마지막 단계로 [11]의 단점을 보완하기 위해서 클래스의 상호관계와 메타모델링(meta-modeling) 기반으로 한 객체와 객체간의 관계, 객체 내부간의 관계를 더 보완한 것이가 버전관리 단계이다. 이 단계는 정보의 상세화와 연결관계에서 구현하기 어려운 양방향 관련성도 쉽게 정의할 수 있게 설계한다. 이 과정은 다음과 같은 과정을 한다. 첫째로는 산출물 객체에 대한 메타 모델링이 이루어져야 하고 둘째로는 산출물 객체간의 관계에 대한 메타 모델링 하여 객체간의 관계를 나타낸 것이다. 마지막으로는 객체 내부에 대한 메타 모델링이 진행되어야 한다. 메타 모델링 결과는 UML을 포함하여 네트워크 기반 객체 지향 개발 방법론의 제반 산출물을 텍스트 형태로 기술할 수 있도록 한다. 또한 버전 관리 단계의 수행과정은 객체지향 소프트웨어 공학 프로세서에 의해 생성되는 제반 산출물을 객체 형태로 통합 관리하는 단계이다. 이 버전 관리 단계는 객체지향 다이어그램에서 표현되는 그래픽 의미를 객체저장소에 저장하여 이 저장소로부터 계층화가 이루어지고, 클래스 부품의 계층화를 통한 분석된 도메인 정보를 손쉽게 추가, 삭제하는 등의 작업으로 소프트웨어 재사용의 효율성을 보이도록 하였다

4.3.1 버전관계에서 객체와 객체간 관계의 메타모델링

클래스정보와 클래스 사이의 관계정보가 객체저장소에 저장된다. 이때 이들 정보는 객체지향 다이어그램 도구를 위한 정보로 변환시켜야한다. 이들 정보는 클래스 정보와 클래스 사이의 관계정보, 기호정보, 위치정보, 다이어그램의 내부 구조 정보를 통합한다.

주요 산출물에 대한 메타 모델링을 수행하는데 크게 세 가지가 있다. 그 하나가 다이어그램이고 들은 양식이며, 마지막으로는 컴포넌트이다. 이들 주요 산출물 객체는 클래스 객체의 서브클래스로 모델링된다. 클래스 객체는 산출물들의 버전관리를 위해 버전 관계(version relationship)를 표현하는데 부모 버전과 아들 버전의 관계를 형성시킬 수 있으며, 연결 관계 정보를 함께 형성한다.

```
abstract class Object
{
    attribute :
        object_id Oid;
        version_number Number;
        parent_version Oid;
        child_version setof Oid;
        relation_version setof Oid;
    method :
        derive() returns Object;
        get_parent_version() returns Object;
        get_child_version() returns Set<Object>;
    get_relation_version() returns Set<Object>;
```

```
    class_method :
        new() returns Object;
}
```

(리스트 1) 버전 관계의 연결 관계표현

클래스는 Product는 산출물을 의미한다. 이 클래스에는 다이어그램, 양식, 컴포넌트에 공통적인 정보들, 즉 설명 속성들이 포함되어 있다. 여기에는 산출물의 번호(id), 종류(name), 설명(description), 연결관계 정보(relation), 작성자(creator), 작성시간날짜(create_time)등의 검색을 위한 속성들이 포함된다.

```
abstract class Product : Object
{
    attribute :
            id Numer;
            name String;
            description String;
            relation String;
            creator String;
            create_time Time stamp;
            file_name char(20);
}
```

(리스트 2) 산출물 관계의 속성들

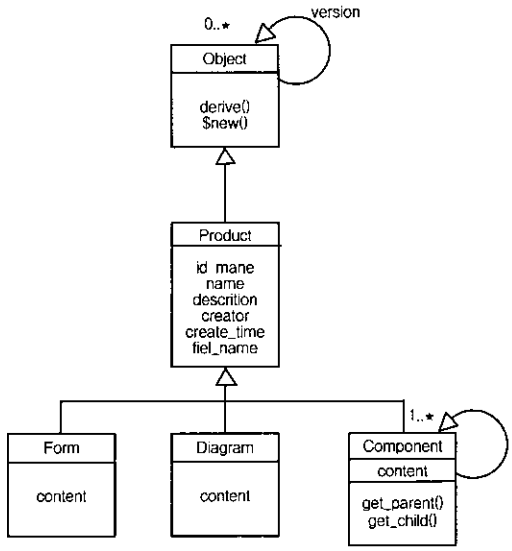
클래스 Diagram은 다이어그램의 원본 내용을, 클래스 Form은 양식의 원본 내용을 클래스 Component는 컴포넌트의 원본 내용을 해석 없이 저장한다. 여기에서는 순수 내용만을 저장하여 설계하기전의 정보를 저장하는 단계이다. 예로서 다이어 그램에는 순수 내용만을 저장하는 것과 관계정보의 내용을 포함하는 다이어그램 관계정보가 있다.

```
class Diagram : Product
{
    attribute :
    content Graphic;
}
class Form : Product
{
    attribute :
        content Text;
}

class Component : Product
{
    attribute :
        content BLOB;
        parent_component Oid;
        child_component setof Oid;
        relation_component Oid
    method :
        derive() returns Object;
        get_parent_component() returns Object;
        get_child_component() returns Object
        get_child_component() returns Object
        Set<Object>;
}
```

(리스트 3) 다이어 그램, 양식, 컴포넌트의 상속 관계

클래스가 컴포넌트 내부에는 컴포넌트의 형상 관리를 위한 속성과 메소드가 포함되어 있다. 속성 parent_component는 해당 컴포넌트를 포함하고 있는 상위 레벨의 컴포넌트를 표현하고, 속성 child_component는 해당 컴포넌트가 포함하고 있는 하위 레벨의 컴포넌트를 표현한다. 이러한 메타모델링 결과를 UML로 표현하면 (그림 9)과 같다.



(그림 9) 객체 관계의 내부메타모델링의 UML 표현

4.3.2 버전관계에서 객체 내부의 메타모델링과 연결관계

객체 내부의 메타모델링에서는 양식이나 다이어그램의 내부 내용 구조를 세부적으로 표현하거나, 특정 양식이나 다이어그램간에 존재하는 종속(dependency) 관계를 표현 할 수 있다. 따라서, 클래스 다이어그램과 상태 다이어그램 사이에는 종속 관계가 성립한다. 즉, 역할과 행위도 포함시킬 수 있으며, 클래스 다이어그램에 속한 각 클래스들은 하나의 상태 다이어그램과 연결되어 있다. 클래스 다이어그램이 삭제되면 앞에서 언급한 역할 포인터와 행위 포인터에 의해서 이에 대한 상태 다이어그램들도 삭제되는데 자동적으로 이루어지도록 설계했다. 이것은 항상 포인터를 저장하고 있기 때문에 가능하다. 또한 PART-OF 관계와 종속 관계가 동시에 성립하고 있는데 이러한 관계는 상태다이어그램으로 성립되고 (리스트 4)와 같이 표현할 수 있다.

```
part of relationship
{       containing : Class_Diagram;
    contained : setof[1..*] State_Diagram dependent;
    contained : setof[1..*] relation_Diagram dependent; }
```

(리스트 4) 증속 관계 표현

5. 적용 사례 및 비교 분석

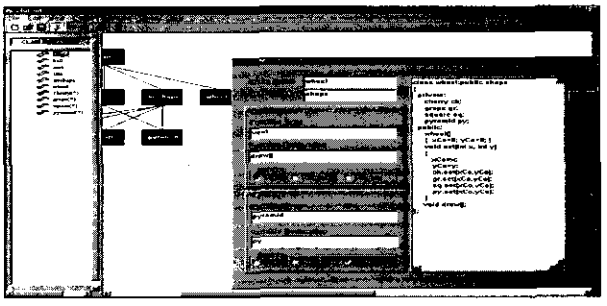
본 연구는 통합 관리 모델을 이용한 효율적인 객체 관리

저장소 설계기법을 제시하고 기존 시스템과 비교 분석하였다. 클래스 부품의 정보분석과 추출 View 기능에 대한 기존 시스템들과의 비교가 부품성격, 부품의 구성항목, 재사용방법, Viewer 기능 등의 특성들을 기준으로 하여 <표 2>에 나타나 있다. 기존 시스템은 데이터, 프로세스, Viewer로 구성되며 클래스 부품, 구성요소, 계층도를 보여주고 재사용 가능한 객체를 분석하여 이들 클래스 정보와 클래스 사이의 관계 정보만을 이용하지만 본 연구의 객체 관리 저장소는 버전 관리와 "연결관계 상호 관련기"를 두어 객체내부 관계와 서로 관련성을 개발자가 쉽게 파악하여 재사용함으로서 클래스 수정환경에서 개발자는 어플리케이션에 관련된 부분만을 개발할 수 있도록 지원하게 설계한다. 또한 기존 시스템에서의 재사용 가능 부품을 다이어그램으로 표현하였지만 본 연구의 객체 관리 저장소는 클래스 다이어그램과 상태 다이어그램 사이에는 종속 관계로 역할과 행위도 포함시킬 수 있으며, 클래스 다이어그램에 속한 각 클래스들은 하나의 상태 다이어그램과 연결되어 있다. 구성 정보를 이용하여 클래스의 생성이 가능하도록 하여 시스템 설계시 그 효율성을 높이도록 설계하였다.

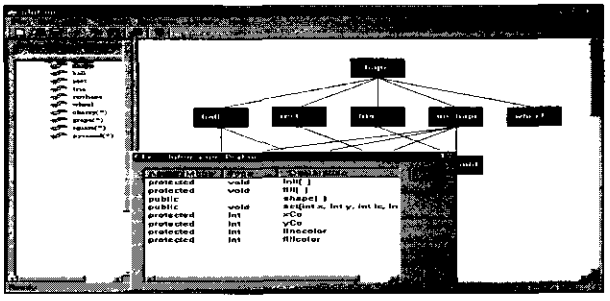
<표 2> 기존의 시스템과 비교

항 목 / 시스템	부품성격	구성항목	재사용 방 법	Viewer
Repository /MVS	함 수	함수형 데이터형	함수호출	추상화 명세화
DDM	클래스	클래스	제한적 라이브러리 제공	×
Teamwork [3]	함 수	함수형 데이터형	×	×
CARS	클래스	클래스	객체의 생성상속	클래스 계층구조
재사용 처리기[9]	함 수	함수형 데이터형	텍스트 명령어를 이용한 재사용	×
객체의 추출과 이해[11]	클래스	클래스 상세정보	객체의 생성상속	클래스 계층구조, 클래스 구성요소, 프로토타입 생성
본 논문의 설계 방법	클래스	클래스 상세정보, 함수형	객체의 생성상속, 버전관리	연결관계 관련기, 객체내부관계 버전관리,클래스, 계층구조,

비교 분석 측면에서 보면 "재사용 처리기[9]"의 재사용 처리기는 수작업을 통한 언어 처리기로 구성되어 있어서 프롬프트 상에서의 명령어를 통하여 재사용 부품을 생성하나, 부품을 명세서로 국한시키기 때문에 실질적인 부품 재사용이라 할 수 없고, 또한 "객체지향 클래스 라이브러리 시스템[11]"은 클래스의 다중 상속에 대한 객체의 이해도가 부족하다. 그리고 Teamwork[3]은 구조적 설계방법과 OOA 기능이 뛰어나지만 재사용성에 있어서는 구조적 분석기법을 통한 프로세스 참조 기능뿐이다. 본 연구는 클래스의 이해를 위한 클래스 계층도의 다중 상속 관계를 표현하였다. 또한 클래스의 다중상속(Multiple Inheritance) 관계를 클래스 사이에 하나이상의 라인 기호로 나타내어 프로그램의 이해도를 높이도록 하고 객체 클래스 내부 계층도에 심볼 "1...*"를 이용하여 프로그램의 구조를 쉽게 이해할 수 있도록 하였다. 클래스 생성(Create) 기능에서는 생성 과정에 프로토타입을 제공함으로서 상속관계 정보를 이용한 클래스 계층도를 쉽게 구성 할 수 있도록 하였다. (그림 11)과 (그림 12)는 클래스 생성과 클래스 구성요소를 나타낸 것이다.



(그림 11) 클래스 생성



(그림 12) 클래스 구성요소

6. 결 론

본 논문에서는 통합 객체 관리 모델을 중심으로 한 객체 관리 저장소 설계 방법을 제시한다. 새로운 소프트웨어 요구시 이전의 유사한 도메인 분석을 통하여 클래스 부품 재사용을 위한 객체 관리 저장소를 데이터(data), 프로세스(process), 뷰(viewer), 버전관리(Version)로 분류하고 프로그램 이해와 재사용을 위한 "연결 관련성 관리기"의 기능을 강화시키고, 객체들의 산출물을 효율적으로 관리 할 수 있도록 하였다.

본 연구에서 객체관리를 효율적으로 관리하고, 객체들의 내부 관계성을 표현함으로서, 기존의 시스템들의 단점인 정보의 표현에 한정되어 있는 것을 극복시킬 수 있었다. 템플리트를 활용했으며, 연결관계가 변화할 때마다 이에 필요한 참조관계를 정의해 주어야 하고, 이전의 참조관계를 삭제하는 일을 프로그래머가 하지 않도록 지원하는 것이다. 기존

의 단점인 객체와 객체간의 구조와 객체간의 내부 모델간의 관련된 정보가 없는것을 객체 내부 메타 모델링하여 개발하는데 필요한 산출물들을 생성, 관리, 검색, 저장할 수 있는 객체 관리 저장소 모델을 제안하고, 설계하는 기법을 제시했다.

본 연구의 향후 과제는 이 모델을 기준으로 클래스 부품의 재사용을 위한 시스템을 개발함으로서 통합 환경에서 부품의 다양한 정보를 이용할 수 있는 웹 재사용 시스템을 개발하는 것이다.

참 고 문 헌

[1] Boehm, B., "Software engineering," IEEE Trans. Comput., Chap.25, No.12, 1976, pp.1226-41.

[2] J. M. Sagawa, "Repository Manager Technology," IBMSystemJournal, Vol.29, No.2, pp.209-227, 1990.

[3] Udo Hahn, Mattthias Jarke, Thomas Rose, "Teamwork Support in a Knowledge-Based Information Systems Environment," IEEE Transactions on Software Engineering, pp.467-481, 1991.

[4] Bernstein, P.A., and Dayal,U., "An Overview of Repository Technology," in Proc. VLDB, 1994, pp.705-713.

[5] James PetroandMichael E. Fotta, "Model-Based Reused Repositories-Conceptsand Experience," IEEE Computer Society Press-Technical Council on Software Eng., pp.60-69, 1995.

[6] Bernstein, P.A., "The Microsoft Repository," in Proc. VLDB, 1997, pp.3-12.

[7] Grady Booch, Ivar Jacobson, and James Rumbaugh. Unified Modeling Language. Rational Software Corporation. January 1997. Version 1.0.

[8] Rational Soft. Corp, http : //www.rational.co.kr/Product/ Rose/.

[9] 김재생, 송영재, "재사용에 기반한 객체들의 정보 분석과 자동화에 관한 연구", 한국정보처리 논문지, 제4권 2호, pp.384-394, Feb., 1997.

[10] 김진수, "모델링패턴을 이용한 연결관계 관련성 관리자", 한국정보처리학회 추계학술발표집, 1998.

[11] 한정수, 송영재, "클래스 부품의 재사용을 위한 객체의 추출과 이해", 정보처리 학회논문지, 제6권 4호, 한국정보처리학회, pp.941-951, 1999.4.

[12] 선수균, 송영재, "통합 객체 관리 모델을 위한 F77/J++ 생성기에 관한 연구", 정보처리 학회논문지, 제7권 제10호, 한국정보처리학회, pp.3064-3074, 2000.11.

선 수 균

e-mail : sksun@Tongwon.ac.kr

1988년 경희대학교 공과대학 전자계산공학과(공학사)

1994년 경희대학교 대학원 전자계산공학과(공학석사)

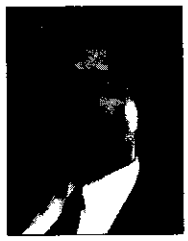
1998년 경희대학교 대학원 전자계산공학과 박사 수료

1988년~1996년 경희대학교 전자계산소 근무

1996년~1997년 영월전문대학 전자계산과 전임강사 및 전자계산소 부소장

1997년~현재 동원대학 사무자동화과 조교수

관심분야 : 소프트웨어 공학, S/W 재사용, CASE 도구



송 영 재

e-mail : yjsong@nms.kyunghee.ac.kr

1969년 인하대학교 전기공학과(공학사)

1976년 일본 Keio University 전산학과(공학석사)

1979년 명지대학교 대학원 졸업(공학박사)

1971년~1973년 일본 Toyo Seiko 연구원

1982년~1983년 미국 Univ. of Maryland 전산학과 연구교수

1985년~1989년 IEEE Computer Society 한국지회부회장

1984년~1989년 경희대학교 전자계산소장

1976년~현재 경희대학교 전자계산공학과 교수

1993년~1995년 경희대학교 교무처장

1996년~1998년 경희대학교 공과대학장

1998년~현재 경희대학교 기획조정실장

관심분야 : 소프트웨어공학, OOP/S, CASE 도구, S/W 개발도구론, S/W 재사용, 컴포넌트 통합 환경