웹을 사용한 객체지향 설계정보 분석
(Object-Oriented Design Object Analyzer based on the WWW)

배 명 남 † 최 완 †† 양 현 택 †††
(Myung Nam Bae) (Wan Choi) (Hyun Teak Yang)

요 약 본 논문은 통합 개발 환경에서 여러 관점으로 작성된 설계 내역들을 효과적으로 분석하기 위한 방법론을 제안한다. 이 방법론은 설계 정보의 역할과 기능을 잘 명세하는 고유의 세부 내역과 관계를 추출하고, 연관된 다른 내역들을 웹 위에서 쉽게 접근하고 파악하기 위한 수단을 제공한다. 이를 위해, 이 방법론은 세부내역을 구분하여 분석 관점의 관계를 정의하는 방법, 웹 객체로 변환하는 방법, 웹 상에서 관계 정보에 따라 관련 내역으로 항해하는 방법을 제시한다.

이 방법론은 다음과 같은 세 가지 장점을 가진다. 첫째, 고안된 설계내역을 다양한 관점에서 분석하는 방식을 제공하며, 둘째, 여러 형식의 연관된 설계정보들을 웹의 하이퍼텍스트 항해 방식에 따라 쉽게 접근할 수 있다. 마지막으로, 웹을 기반으로 각 개발 방법론 내 설계내역의 고유 표현 방식과 독립적인 표현 수단을 사용함으로써 원격지 설계내역에 대한 공유 및 접근이 용이하다.

Abstract This paper proposes a methodology for effectively analyzing a lot of design information written from various design viewpoints in an integrated s/w development environment. It provides a way of extracting the inherent role of the design contents, as well as easily accessing and understanding the other related ones on the web. For this aim, it defines a series of methods 1) to correctly capture relationships between the contents and the other related ones, 2) to translate them into a web object equipped with hypertext links corresponding the relationships, and to navigate all the related contents with the links.

The suggested methodology have three advantages. First, it supplies ways to analyze new design contents invented from different points of view. Second, it can help user to easily access the related contents using hypertext links over WWW. Third, it provides an environment capable of effectively sharing and accessing design contents distributed on remote sites, thanks to the platform-independent property of the web.

1. 서 론

복잡한 대형 시스템의 설계는 시간과 신뢰도에 관한 엄격한 제약 조건하에서 주어진 역할을 명세하는 복잡한 작업이다. 특히, 컴퓨터 통신과 같이 외부와 밀접한 관계를 맺고 있는 응용 영역은 외부 환경과의 상호작용에 대한 신뢰성이 요구된다. 따라서, 신뢰성 있는 시스템의 설계를 위한 많은 정형화된 방법들[4, 18, 19]이 연구되었다. 그런데, 최근 들어, 시스템이 보다 대형·복잡화되면서 시스템의 설계 뿐 아니라 유지 보수 및 소프트웨어의 재사용과 같은 측면도 동시에 지원하기 위한 수단이 요구되고 있다.

재사용은 소프트웨어 개발에 있어서 생산성과 품질 향상에 중요한 역할을 한다. 한 예로, 현재 음성 서비스용 교환기 시스템(switching system)은 B-ISDN, 지능망 등의 서비스가 계속 추가되고 있으며, 최근에는 이동통신 서비스가 가세함에 따라 점점 더 복잡/다양해지고 있다. 이때, 기존 시스템내에는 재사용 가능한 핵심적인 부분들이 많이 존재하고, 이들 중 많은 부분은 새로운 시스템에 큰 변화없이 재 사용될 수 있다. 이를 위해서는, 재사용 가능한 자원을 구분하고 재사용을 위한 체계

† 정 회 원 : 한국전자통신연구원 실시간DBMS팀 연구원
mnbae@etri.re.kr
†† 비 회 원 : 한국전자통신연구원 실시간DBMS 연구원
wchoi@etri.re.kr
††† 비 회 원 : 순천대학교 컴퓨터과학과
htyang@moinvalley.co.kr
논문접수 : 1999년 10월 18일
심사완료 : 2000년 5월 26일

적인 분석 수단을 제공하여야 하는데, 이 경우, 재사용 가능한 코드와 설계 내역들을 서로 연관시키고 참조할 수 있는 자동화된 도구는 필수적이다. 이 도구들은 명세 과정에서의 편리성을 제공할 뿐만 아니라 개발자의 실수를 방지하고 그 명세에 따라 설계내역을 검증하는데 도움을 준다.

한편, 현재의 소프트웨어 개발환경은 점차 분산 환경 위에서 이종 언어 및 다양한 플랫폼을 지원하고 있기 때문에, 재사용 가능한 자원들은 대부분의 경우 네트웍상의 이종 플랫폼 기반의 서버에 분산된다. 따라서, 이 자원들의 분석 및 재적용을 위해서는 네트웍상에 분산된 각 자원들을 하나의 일관된 체계로 접근하고 추적하기 위한 환경이 필요하다. 웹은 분산 환경에서 여러 형식의 자료를 표현하고 전송하기 위한 표준화된 방식을 제공하며, 또한 하이퍼텍스트 방식을 통해 정보를 쉽게 조직화하고 참조할 수 있는 기본 구조를 지원하고 있어 정보의 공유와 전달에 효과적인 수단으로 사용되고 있다. 따라서, 본 논문에서는 여러 곳에 산재되어 있는 설계내역들간의 참조를 위한 방법으로 웹을 활용하고자 한다.

본 논문에서는 통합 개발 환경에서 얻은 각종 내역들에 대해, 내역의 분석 및 재적용을 위한 각종 분석정보들을 생성하고, 이들을 웹위에서 참조할 수 있도록 하는 방법에 대해 기술한다. 이를 위해, 기존에 우리가 고안된 바 있는 설계정보 저장구조[22]를 새로운 서비스에 맞는 저장소로 확장하고, 저장소내의 각종 정보들을 개발자에게 보이고 요구에 따라 분석 정보들을 제어하기 위한 체계를 정립한다.

본 논문의 2장에서는 관련 연구로 설계정보의 분석을 위한 도구들에 대해 요약하고, 3장에서는 웹위에서 서비스하게된 동기와 이에 필요한 저장소와 접근 방법에 대해 선행 연구와 비교하여 기술하고, 4장에서는 객체지향 설계정보를 실제 웹정보들로 변환하기 위한 각종 규칙과 방법에 대해, 5장에서는 결론 및 향후 연구과제에 대해 기술한다.

2. 관련연구

객체지향 소프트웨어의 분석과 이를 통한 재사용 수단을 제공하려는 많은 도구들과 제반 환경들이 소개되었다. 하지만, 현재까지의 연구는 주로 코드 중심의 환경에서 이루어져 왔으며, 코드이외에 보다 추상적이고 재활용에 적합한 다른 설계 내역의 활용 방안에 대해서는 고려하지 않았다는 문제점이 있다. 그 주된 이유중의 하나는 각 개발 단계 중에 생성되는 설계 내역들간의 일관성 유지 및 자동화된 변환 방법을 지원하기 어려웠기 때문이다. 현재의 CASE 도구와 통합 환경은 범용화된 응용에 적용되도록 정의되었기 때문에, 이러한 변환을 충분히 제공하지 못한다. 반면에, 통신 분야와 같이 운용 환경이 제한된 응용에서는 그 연산이 단순하고 명료하기 때문에, 제한된 설계 명세에서 구현 언어로의 변환이 쉽게 가능하다[4, 19]. 이 연구들은 통합화된 개발 환경을 구축하기 위해, 주로 특정 응용 영역에 적합하도록 초기 단계의 모델링 기능에 제약을 가하고, 이후 단계에서는 정형화된 방법을 사용하여 자동화된 변환을 제공하고 있다. 즉, 분석단계에서 설계단계까지의 전반부는 [8, 17]와 같은 범용 개발 방법론으로 내역을 작성하며, 후반부에서, 이들은 [5, 14]와 같은 정형적인 방법론을 통해 변환되어 검증된 후 자동적으로 코드를 생성하는 과정으로 이루어진다. 이와 같이 통합화된 개발 환경에서는 각 개발 단계별 설계 정보의 변환이 일관성있게 이루어지기 때문에, 설계 정보 기원으로의 일관된 추적과 시험 및 적용이 가능하게 된다. 이때, 통합 환경은 변환 과정상의 제반 정보들을 관리한다. 본 논문에서 제안하는 저장소와 관련 도구들은 이러한 제반 정보를 바탕으로, 요구 분석에서 구현까지의 각 단계에 존재하는 연관된 내역들을 충분히 파악한 후, 웹을 통해 이들에 대한 참조 수단을 제공한다. 즉, 통합 환경에서 분석하여 얻은 각종 내역들을 웹 위에서 참조하고, 해당 내역의 분석과 이해를 위한 여러 분석적 수단을 제공하기 위한 절차와 방법을 기술한다.

현재, 구축된 소프트웨어 내역을 분석하여 사용자가 특정 내역에 쉽게 접근하고 분석하기 위한 여러 표현적 방식을 갖는 분석도구들이 제안되어 사용되고 있다. 먼저, 구현된 내역의 분석을 위한 도구로 cscope[7], pcGrasp[15]와 같은 도구들이 많이 사용되고 있다. 이들은 구현 내역에 대한 부분적인 파싱을 통해, 함수 정의나 호출 혹은 전역 변수 대상 검색 등과 같은 내역 분석을 위한 여러 방식을 제공하고 있다. 하지만 이들은 내역의 체계적인 분석을 위해 코드와 동등한 의미를 갖는 코드이외의 다른 내역들은 활용할 수 없다는 단점이 있다. 보다 진보된 분석 방식을 제공하는 Hyperbook [11], PEEL/CodeFinder[10], OO-Browser, xlint++와 같은 방식은 코드 자체에 대한 정보뿐만 아니라, 이의 분석을 통해 보다 유용한 정보를 제공한다. 예를 들어, 클래스 계층구조와 같은 설계관점의 뷰를 구성하여 그래픽하게 제시하고, 이 뷰에서 관련된 다른 정보로의 항해를 제공한다. 하지만 이 도구들이 제공하는 뷰내의 내용은 시스템 전체적인 관점에서 재구성된 것으로, 초

기 설계단계에서의 개발자가 작성한 관점과는 일치하지 않는다는 단점이 있다. 즉, 설계 단계에서 고안한 내역들은 단위 내역(한 설계정보)별로 고유의 역할과 기능을 명세하기 위한 관점에서 작성됨에도 불구하고, [11]가 제공하는 뷰는 이러한 관점을 그대로 유지하지 못한다. 개발자에게 보다 유용한 정보는 특정 내역이 각 개발 단계에서 어떠한 역할로 사용되고 있으며, 최종적으로 코드내에서 어떻게 표현되고 있는가에 대한 메타 정보라고 할 수 있다. 특히 대형 시스템과 같이 복잡도가 높으면서도 재적용율이 높은 분야에서는 이러한 정보의 추출 및 참조가 필수적이라 할 수 있다. CASE 도구의 측면에서, Argo/UML[21]은 항해를 위한 일련의 기본 규칙들을 제공하고, 이들의 조합을 통해 한 내역에서 관련된 다른 내역으로의 항해를 제공하고 있다. 하지만, 이 도구는 notation에 대한 추가의 의미 파악 수단을 갖고 있지 않아, 단지 구조적으로 추출가능한 내역들간의 항해만을 제공한다는 제약이 있다. 즉, 동일한 시멘틱을 가지며 다른 다이어그램내에 존재하는 내역들에 대한 상호 참조나 이를 바탕으로 한 보다 상위 레벨의 정보(다이어그램)로의 참조를 위한 수단은 제공하지 못하고 있다.

다른 측면의 관련 연구로서, *GoFigure* [18]나 *More* [9] 등은 방대한 정보들을 구조화하고 이 정보를 효율적으로 서비스하기 위해, 1) 데이타베이스 시스템이 제공하는 데이타 일관성 유지, 회복, 쿼리(query) 등과 같은 기능들을 기반으로 하고, 2) 이 시스템을 웹에 연동하여 사용자는 웹 브라우저를 통해 질의하고, 그 결과를 다시 웹을 통해 볼 수 있도록 하고 있다. 이러한 방식은 빠른 시간내에 플랫폼 독립적인 GUI 개발을 위한 환경을 제공하며, 방대한 자료에 대한 다양한 데이타베이스 서비스를 제공할 수 있다. 이러한 방식을 사용한 분석 환경은 원거리에 분산되어 있는 여러 설계내역들을 확인하고 분석한 후 지역 서버내에 다운(down)받아 구축중인 자신의 설계 작업에 활용할 수 있으며, 향후 분산화된 개발 환경에서의 다양한 서비스 확장에도 용이하다는 장점이 있다.

3. 웹 환경에서의 설계정보

분산 환경에서 설계정보를 분석하고 이해하기 위해서는 설계 정보 자체 뿐만 아니라 네트웍상의 다양한 플랫폼과 독립적인 구조에 대해 고려하여야 한다. 웹은 높은 호환성과 데이타베이스 시스템이 제공하는 유용한 데이타 관리 기능을 통해 안정성과 범용성을 동시에 만족시킬 수 있는 해결책으로[11], 이러한 문제점들을 쉽게 해결할 수 있는 구조적 특성을 가지고 있다. 또한 하이퍼텍스트 구조를 통해 전체적이고 개념적인 내역에서 보다 상세한 내역으로의 점진적인 분석이 가능하며, 접근한 대상에 대한 이력(history) 유지 기능 등과 같은 사용자 인터페이스를 함께 제공한다는 장점이 있다[1]. 이러한 특징은 본 논문의 주요 내용인 소프트웨어내의 내역들간의 항해 구조를 가장 잘 적용할 수 있는 기반 환경을 제공하고, 여러 형태의 설계내역을 별도의 뷰어 없이도 용이하게 나타낼 수 있도록 한다. 따라서, 우리는 기존의 고안된 설계정보 저장구조[22, 23]가 웹이 갖는 이러한 장점들을 수용하도록 확장한다. 즉, 대상 소프트웨어 시스템의 개발 과정에서 얻은 메타 정보들을 데이타베이스 내에 유지하고, 웹을 통해 유용한 서비스를 제공함으로서 보다 거리/환경적인 제약에 독립적인 소프트웨어 분석 환경을 제공하고한다. 이를 위한 구체적인 방법으로, 설계 도구에서 얻은 정보들이 웹 환경에 보여지고, 이를 운영하기 위해 필요한 일련의 정보들로의 변환 규칙과 운영 방법에 대해 설명한다(자세한 내용은 4장에서 설명). 이러한 요구는 그래픽 표현을 기본으로 하는 객체지향 개발 방법론의 보급으로, 1) 설계내역을 보이기 위해 여러 미디어의 표현 방식이 필요하고, 2) 표현된 설계내역에 대한 연관성 추적, 그리고, 3) 하드웨어 및 운영체제 플랫폼에 독립적인 환경이 필요하기 때문에 대두되었다.

본 시스템의 구성은 이미 구축된 각종 설계정보 저장구조를 웹에서 인식할 수 있도록 하기 위한 저장소와 이들을 운용하기 위한 관련 도구들로 이루어진다. 대략적인 시스템의 구조는 그림 1과 같다.

객체 분석기는 고안된 각각의 설계 정보들을 유일하게 구분하는데 필요한 메타 정보(항해 식별자)를 생성한다. 얻어진 정보는 특정 설계 내역의 위치를 같이 명시하기 때문에, 그 설계 내역으로의 참조를 위해 사용되며, 사용자는 참조를 통해 해당 설계내역을 사용하는(혹

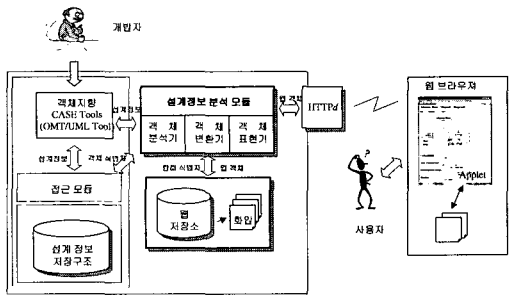

그림 1 시스템 구성

은 사용되는) 등 여러 관점으로 파악할 수 있다. 객체 변환기는 분석기가 생성한 메타 정보와 설계정보들을 웹 객체로 변환하여 웹 저장소에 저장하고, 이후 사용자 요구에 따라 해당 객체를 추출한다. 이때, 요구된 정보가 웹 저장소 내에 존재하지 않을 경우, 설계정보 저장 구조로의 접근을 통해 얻는다. 객체 표현기는 저장소에서 추출된 설계정보들의 형태(다이어그램/코드 혹은 인스턴스의 집합 등)에 따라 여러 시각화 수단을 제공한다. 주로 웹 객체들을 웹 브라우져내에 나타내기 위한 여러 규칙들의 모임이다.

[22, 23]는 CASE 도구를 사용해 작성한 설계 내역을 저장하고, 단위 설계 정보들간의 구조적인 관계를 파악하기 위한 구조와 모델이다. 그러나, 이들로부터 얻은 설계 정보들을 웹위에서 분석하기 위해서는 관련 내역의 추출 뿐만 아니라 웹 환경에서 요구하는 일련의 규약들을 만족시켜야 한다. 즉, 개발자 환경(웹 브라우져)에서 특정 설계 내역에 접근하고, 다시 선택된 세부내역이 웹을 통해 시각화될 수 있는 절차와 방법이 제공되어야 한다. 이를 위해, 먼저 웹을 통한 접근 단위에 대해 고려해 보자. 통합 개발환경에서 단위 모듈을 개발한 결과는 그림 2와 같이 개발 단계에 따라 일정한 연관성을 가진 다수의 설계 내역들로 존재하며, 이들은 다시 설계 내역의 보다 세세한 의미 혹은 기능적인 역할을 명세하는 구체화된 세부 내역들로 구성된다.

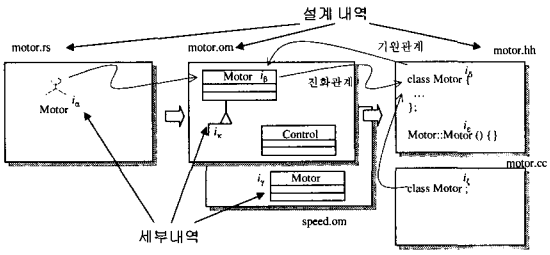

그림 2 여러 개발 단계에 산재된 내역들

그림 2에서 설계 내역 'motor.rs'내 액터 i_α는 설계단계의 클래스인 i_β의 이전 단계 내역이며, 다시 i_β를 거쳐 구현 코드 내 i_δ, i_ε 등으로 변환됨을 알 수 있다. 또한, i_ζ은 i_δ를 참조하고 있다. 이와 같이 다른 내역과 관계를 가짐으로써, 시스템에서 항해의 대상이 되는 내역을 '참조가능한 내역(RC)'이라고 정의한다. 즉, RC는 소프트웨어 시스템내에서 이전내역에서 진화하였거나, 혹은 다음 내역의 기원이 되는 모든 내역을 지칭한다.

이때, 참조 가능한 내역들은 여러 설계 내역에 중복되어 명세될 수 있지만, 분석적 측면에서 그 형상은 그대로 유지되어야 한다. 예를 들어, 'motor.om'내 내역 i_β, i_γ는 'Motor' 클래스에 대해 중복해서 명세하고 있지만 이들은 각기 다른 표현적 역할을 추가로 갖고 있다. 즉, i_β는 i_κ에 대해 기반 클래스(base class) 역할이 함께 명세된 반면, i_γ는 그렇지 않다. 이러한 역할은 이후 i_β가 참여하는 모든 관계성을 포함하는 객체도 설계내역의 추출과 같은 연산에 유용하게 활용될 수 있다. 예를 들어, 'Motor' 클래스를 포함하는 모델을 검색하라는 요구에 대해 'motor.om'과 'speed.om'이 검색되겠지만, 'Motor' 클래스의 유도 클래스(derived class)를 직접 명세하고 있는 모델의 검색에는 'motor.om'만이 추출될 것이다. 또한, 이러한 형상의 유지는 [11]과는 달리 내역을 고안한 설계자의 의도가 일련의 변환 과정에서 삭제되지 않고 재적용하고자 하는 개발자에게 그대로 보여지도록 하기 위해서도 필요하다.

이처럼, 설계내역의 고유한 설계 의미(design semantics)를 유지하고 여러 분석적 측면의 요구를 만족시키기 위해, 분석 시스템은 그림 3과 같이 개념적으로 두 계층의 저장소로 구성된다.

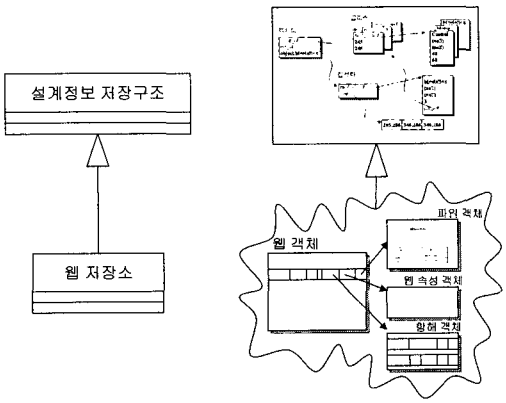

그림 3 설계 정보에서 웹 정보로의 추상화

설계정보 저장구조[22]는 CASE 도구에 의해 작성된 실제 설계 내역들을 저장하고 유지한다. 이 구조는 사용자가 작성한 설계내역의 고유한 특성(위치, 종류, 색, ...)들과 설계 내역들간에 존재하는 구조적인 관계성들을 데이타베이스화하여 저장한다. 반면에, 웹 저장소는 내역의 분석적 측면의 요구로, 내역의 기원이나 진화와 같은 관계를 파악하고 이에 대한 접근 수단을 제공하기 위한 역할을 담당한다. 다시 말하면, 웹 브라우져를 통

해 참조가능한 내역들을 시각화하고 항해하기 위해 필요한 정보들을 유지하도록 구성되었다. 웹 저장소는 [22]의 하위 구조로 정의되었기 때문에, [22]에서 설계 내역들간의 관계성 파악과 세부내역 검색을 위한 연산들을 상속받는다. 따라서, 웹 저장소는 설계내역의 버전 진화 등과 같은 관계성 파악과 특정 클래스의 세부 속성(attribute)들의 추출과 같은 기능들을 함께 제공할 수 있다. 이러한 구성은 유용한 정보를 단순히 데이타베이스화하고 서비스하는 것과는 달리 향후 데이타 모델의 진화에도 쉽게 적응할 수 있다는 특징을 갖는다.

단위 객체도와 같이 독립적인 설계의미를 갖는 설계내역은 보다 구체화된 의미를 갖는 세부내역들(클래스나 관계성)의 모임으로 이루어진다. 분석 환경에서 설계내역과, 보다 특성화된 단위이며 참조가능한 세부내역을 유일하게 구분하기 위해 다음과 같이 항해 식별자 N을 위한 스킴(부록 A의 NavigationIdentifier 참조)을 정의한다.

참조가능한 내역의 식별을 위한 스킴 = (d′, d″)
d′ ∈ 설계내역을 명시하는 식별자들 집합
d″ ∈ 설계내역내 참조가능한 세부내역의 식별자들 집합 ∪ { ε }

여기에서, d′는 시스템에서 단위 설계내역을 유일하게 구분하며[23], d″는 d′에 의해 구분된 설계내역내의 세부 내역을 유일하게 구분하기 위해 정의하였다. 따라서, 참조 가능한 세부 내역은 항해 식별자를 통해 시스템내에서 유일성을 보장받는다. 여기에서, n∈N에 대해 n.d″=ε이라면, n은 세부 내역이 아니라 n.d′에 의해 구분된 설계내역임을 의미한다. 항해 식별자 N내에 d′는 웹 객체에서 설계정보 저장구조내에 저장된 원래 설계 객체와 연계하기 위해 필요하다. 예를 들어, 객체도 웹 객체의 항해 식별자 n_1∈N에 대해, 사용자가 n_1의 다음 OMT 관점 혹은 이전 버전을 요구하는 경우, 검색의 대상인 n_2는 [23]의 연산 $F_{NEXT_OMT}(n_1.d')=n_2 \in N$를 통해 얻게 된다. 이때, n_2는 n_1의 동적 관점을 나타내는 동적도 설계내역(이때, $n_2.d''=\varepsilon$)이다. 이후 웹 저장소에서 파악된 n_2가 지칭하는 웹 객체(설계 내역)를 추출하여 웹 브라우저에 보이게 된다.

이와 같이 계층화된 구조를 갖게 한 이유는 1) [22]과 [23]에서 정의한 유용한 연산들을 재사용하고, 2) 설계정보 저장구조에서 웹 서비스로 인해 발생하는 부담(load)을 웹 저장소로 분산시키며, 또한, 3) 웹을 통해 설계정보의 초기 분석 과정에서 주로 요구되는 전체적인(포괄적인) 개념의 정보들에서 보다 상세한 세부 정보로의 단계별 접근을 위한 여러 수단들을 저장 구조와 독립되게 구성하기 위해서이다. 이러한 구조를 갖지 않는 Rational Rose나 Paradigm Plus와 같은 범용 설계 도구들의 사용을 통해서도 기본적인 설계 내역의 파악 및 분석이 가능하다. 그러나 이들은 고안된 소프트웨어의 분석을 위해 필요한 세부내역에 대한 추적 및 단계별 연관성 파악, 그리고 융통성있는 GUI 기반의 분석 환경등을 제공하지 못하고 있다. 즉, 이들 도구들이 갖는 참조는 모델의 분석 측면이라기 보다는 모델의 일관성(consistency)을 강화하기 위한 방법들이기 때문에, 실제로 모델을 작성한 후 완성된 모델을 재사용하고 분석하기 위한 관점에서 매우 제한적이다. 반면에 본 방식은 웹과 연동된 구성을 통해 이러한 문제점들을 해소하고 있으며, 해당 CASE 도구 없이도 추상적인 단계의 설계 내역의 검색과 분석도 가능하게 한다. 이 구조의 장점은 분산 환경에서 해당 CASE 도구 없이도 원거리에 있는 설계내역을 확인하고 분석한 후 지역 서버내에 다운받아 구축중인 자신의 설계 작업에 재 사용할 수 있는 수단을 제시한다는 점에서 기존의 검색 도구나 분석 지원 도구들[10, 11, 15, 22]에 비해 보다 융통성있는 환경을 제공한다.

4. 저장소 관리 모듈들

웹을 통해 설계내역을 참조하고 분석하기 위해, 설계 도구로 고안한 내역들은 웹 저장소내의 객체로 사상되어야 하며, 이 과정에서 관련 객체들간의 참조 관계를 함께 파악하여야 한다. 또한, 분석된 웹 객체들을 웹 브라우저를 통해 표현하기 위한 수단들이 함께 제공되어야 한다. 이 장에서는 이를 위한 모듈들로 객체 분석기와 변환기, 그리고 표현기에 대해 설명한다.

4.1 객체 분석기

설계 도구를 사용하여 고안한 내역은 최종적으로 구현 코드로 변환되는데, 코드 이전 단계의 객체도 내역은 구현과 관련된 세부적인 내용을 포함하지 않고 보다 개념적이고 전체적인 관점에서 내용을 기술하고 있기 때문에, 초기 분석 단계에서 유용하게 활용될 수 있다. 이를 위해, 분석기는 서로 다른 단계(설계단계와 구현단계와 같이)와 형태(그래픽 다이어그램과 프로그램 코드)로 추출된 내역들을 연관시키기 위해 메타 정보(항해 식별자 등)를 생성한다. 즉, 각 설계 내역내에 포함된 모든 참조가능한 내역에 대해, 이들을 유일하게 구분하고 내역간의 참조 관계들을 파악하기 위한 기본적인 수단을 제공한다.

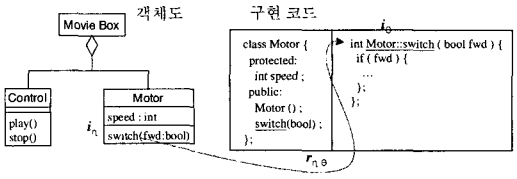

그림 4 객체도에서 구현 코드로의 참조

객체지향 소프트웨어의 개발 과정을 살펴보자. 요구분석 단계의 내역은 최종적으로 구현 단계의 내역으로 변환될 것이다. 이때, 소프트웨어를 분석하고 이해하는 측면에서, 내역을 분석하는 개발자에게 특정 내역이 각 단계별로 갖는 기능적 역할을 제시하고 또한 이후 다른 설명적 관점을 가지는 각각의 진화된 내역들에 쉽게 접근할 수 있게 하는 수단은 매우 유용하다. 예를 들어, 그림 4에서 객체도내 세부내역 'switch(fwd:bool)'(=i_η)와 구현 코드내 세부 내역 'Motor::switch(bool fwd)'(=i_θ)간에 진화 관계가 설정되어 있다면, 내역 i_η의 설계 단계 뷰를 i_θ에서 인식할 수 있다.이와 같은 방식은 복잡한 시스템의 한 세부 내역을 각기 다른 관점에서 파악할 수 있어, 해당 내역에 대한 이해도와 재적용률을 향상시킬 수 있다.

우리는 이러한 연관성을 내역간의 관계로 정의한다. 관계의 종류는 객체도에서 코드와 같이 다음 개발 단계 내역으로의 발전을 나타내는 '진화' 관계, 코드내 메소드 선언부에서 정의부로의 '정의' 관계 등이 있다. 이때, 항해 식별자의 정의에 따라 모든 참조가능한 내역 i_k, $i_l \in$ RC의 항해 식별자 n_k, $n_l \in$N에 대해, $i_k \neq i_l$이면 $n_k \neq n_l$이고, 만일 $i_k=i_l$이라면 $n_k=n_l$을 항상 만족한다. 따라서, 두 내역간의 관계 R를 내역의 항해 식별자를 사용하여 정의한다(부록 A의 RelationshipTriple 참조).

참조가능한 내역들의 관계 트리플 t = < r, n, n'>

여기서, n, n'∈N는 내역 c, c'를 각각 구분하는 항해 식별자이며,

r 은 내역 c, c'간에 가능한 관계 ∈ {'진화', '정의', ...}

여기에서 관계는 참조 가능한 내역들간에 참조하는 관점을 의미한다. 예를 들어, 그림 4에서 내역 i_η는 보다 구체화된 내역인 i_θ로 진화의 관점을 가져, 관계 트리플 $t_{\eta\theta}$는 <진화, n_η, n_θ>로 표현될 수 있다. 객체분석기는 클래스(class)나 상태(state), 액터(actor)와 같이 참조 가능한 내역을 새로이 생성할 때, 내역의 항해 식별자를 얻고 다른 내역과의 관계를 파악한다. 이를 위해, 객체분석기는 항해 식별자 $n_s \in$N을 가진 내역과 r' ∈R 관계에 있는 ti라는 이름을 가진 내역의 항해 식별자를 얻기 위한 함수 $F_{r'}$를 사용한다.

$F_{r'}$(n_s, ti) → n_d, 여기에서 n_s, $n_d \in$N, r'∈R

ti는 n_d에 의해 지칭되는 세부내역의 명칭(사용자 뷰)

여기에서 n_d는 새로이 생성될 세부 내역의 항해 식별자이며, r'는 항해 식별자 n_s, n_d가 지칭하는 두 내역간 관계의 종류를 명시한다. 예를 들어, 그림 4에서 객체도내의 'switch'에 대한 항해 식별자를 n_η라 하였을 때, n_η에 대해 진화의 관계가 있는 구현 단계의 뷰 'Motor::switch'를 갖는 항해 식별자는 $F_{진화}(n_\eta$, "Motor::switch")=n_θ로 얻고, 여기서 i_η와 i_θ간의 진화 관계(즉, <진화, n_η, $F_{진화}(n_\eta$, "Motor::switch")=n_θ>)가 설정된다(부록 B의 RelationshipTriple 사례 참조). 이후 객체 표현기는 n_θ을 사용하여, i_η에서 i_θ을 참조할 수 있다.

그림 5는 설계/구현 단계간에 설계 정보 사상시, 객체분석기에 의해 생성된 항해 정보의 한 예를 보이고 있다. 참조 가능한 내역들간의 연관성을 나타내는 여러 종류의 관계가 있으며, 개발자에 의한 추가의 정의도 가능하다. 이러한 체계는 [10, 11]과 달리 하나의 내역에서 명시된 다수의 관계에 따라 여러 내역으로의 참조가 가능하다는 장점을 가진다.

한편, 고안되는 많은 설계 내역은 객체도, 동적도와 같은 그래픽화된 정보들이다. 객체 분석기는 참조 가능한 그래픽화된 내역들을 웹에 표현하고 이로부터 관련된 다른 내역으로의 항해가 가능하도록 하기 위해, 내역의 좌표계 정보들을 추가로 추출한다. 이들은 객체 표현기가 설계 내역에 대한 이미지 맵[3]을 구성할 때 하이퍼텍스트 항목으로 사용된다. 예를 들어, 그림 5의 객체도에서 'Motor' 클래스 X 좌표계인 (462, 353, 393, 80

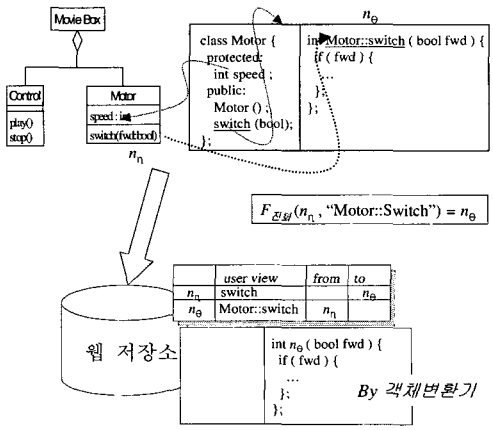

그림 5 항해 정보의 추출

)는 이후, 객체도에 대응해 만들어질 이미지 맵에서 'Motor' 클래스의 구현 내역('Motor::switch')에 대한 참조를 위한 앵커(anchor)로 사용될 수 있다(부록 B의 RefObjectID 사례 참조). 이에 대한 자세한 내용은 4.3 절에서 설명한다.

이와 같이, 객체 분석기는 참조가능한 내역들을 구분하고, 항해 식별자를 부여하며, 이후 웹의 하이퍼텍스트 링크로 변환하기 위해 필요한 각종 관계들을 추출한다. 이 정보들은 웹 저장소내에 저장되며, 다른 설계정보와 함께 조합되어, 웹을 통해 사용자에게 소프트웨어 분석을 위한 정보로 제공된다.

4.2 객체 변환기

웹 저장소는 [22, 23]를 보다 특성화한 내역으로, 내부적으로는 웹 브라우져 상에서 보이기 위한 다수의 파일들과 데이타베이스내 정형화된 정보들을 모아 웹 객체로 다루도록 구성되었다. 이러한 방식은 [22]내의 설계정보가 객체 표현기를 통해 브라우져에 잘 표현될 수 있는 특성을 추가하기 위해 필요하다. 객체 변환기는 설계정보 저장구조와 웹 저장소간의 일관성을 보장하고, 분석된 설계내역을 웹에서 서비스하기 위해 웹 객체로 변환하며, 웹 저장소에서 저장구조[22]로의 접근 인터페이스를 제공한다. 객체 변환기를 통한 저장소의 변경은 반드시 설계 정보 저장구조에 대한 변경 트랜잭션 내에서 수행되기 때문에 설계정보 저장구조와 저장소간의 설계정보에 대한 참조 무결성은 항상 유지된다.

객체 변환기는 객체 분석기에서 얻은 정보를 일관되게 저장·유지하기 위한 일련의 규칙들이다. 그림 6은 객체도로부터 C++ 프로그램 코드가 자동 생성되고, 객체 변환기가 이를 웹 객체로 사상한 내용을 보이고 있다. 웹 객체는 하이퍼텍스트 링크로 대치하기 위한 항해 식별자를 갖도록 변환되고 있다(부록 A의 WebObject와 하위클래스인 ClassDiagramObject 참조).

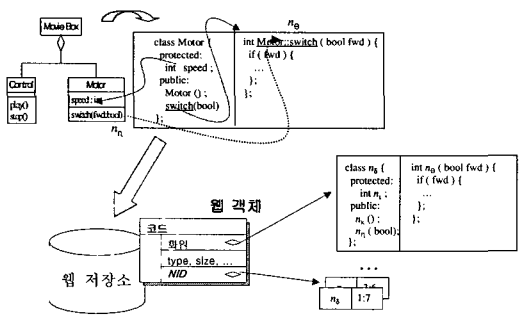

그림 6 웹 객체의 생성

이후 웹 객체는 객체 표현기를 통해 항해 구조를 포함한 웹 문서[2]로 재구성된다. 앞 절에서 기술한 바와 같이 웹 저장소는 설계 내역을 웹 상에서 서비스하기 위해 필요한 정보들만을 가지고 있다. 예를 들어, 그림 6과 같이, 코드 내역에 대응되는 웹 객체는 참조 가능한 모든 내역들에 대한 항해 식별자를 포함하고 있는 코드 파일을 가지고, 추가로 이 파일의 형식, 크기 등과 같은 정보를 함께 갖는다. 하지만, 웹 객체는 자신의 이전 버전과 같은 데이타 모델 의존적인 내용을 포함하지는 않는다. 만일, 사용자가 현재 웹 객체에 대해 이러한 관계성을 가진 다른 웹 객체를 요구한다면, 실제 관계성은 [23]에서 파악되며, 파악된 웹 객체는 웹 저장소에서 추출하여 제공된다. 이를 위해, 변환기는 설계정보 저장구조와의 연동을 위한 접근 인터페이스가 필요하다. 예를 들어, 설계정보 저장구조내 객체도 설계내역의 항해 식별자가 $n_1 \in N$이라 할 때, n_1의 다음 버전 설계내역은 연산 $F_{NEXT_VERSION_VIEW}(n_1.d') = n_2 \in N$로 얻을 수 있다. 저장소는 미리 정의된 인터페이스를 통해 사용자 질의를 전달하고, 웹 위에서 결과로 얻은 n_2가 지칭하는 웹 객체를 추출하여 사용자에게 표현하는 단계로 이루어진다. 앞에서 설명한 바와 같이 항해 식별자는 참조 가능한 내역을 유일하게 구분하기 위해 사용한다. 항해 식별자는 [23]등에서 정의한 객체 식별자와는 다르다. 항해 식별자는 설계정보 저장구조와 웹 저장소, 사용자의 웹 브라우져 환경에서 각각 표현되는 내역을 하나의 체계로 명시하는데 사용된다. 즉, 항해 식별자는 두 저장소에 분산되어 저장된 한 내역을 추출하고 일치시키기 위해 사용되며, 또한 하이퍼텍스트 앵커(anchor)로 변환되어 웹 브라우져내에서 항해를 위해서도 사용된다.

4.3 객체 표현기

객체 표현기는 저장소내의 각종 객체들을 웹 브라우져에 표현하기 위한 웹 문서를 생성하는 일련의 규칙들로, 크게 1) 각종 웹 객체를 웹 브라우져가 인식할 수 있는 표현으로 변환하며, 참조가능한 내역에 대해 하이퍼텍스트 정보 형태로 구성하는 그룹과, 2) 개발자의 선택에 의해 해당 웹 객체와 구조적인 관계성을 가진 관련 설계 정보를 추적하기 위한 그룹이 있다. 전자의 경우는 주로 웹 객체의 표현과 관련되어 있으며, 후자의 경우 [22, 23]에 접근하는 연산이다. 연산의 예는 프로그램 코드내 내역들간의 참조 관계(항해 식별자)를 하이퍼텍스트 링크로 재구성하고, 객체도와 같은 그래픽 정보를 시각화하며 이미지 맵을 구성하는 기능, [22]의 검색 결과로 여러 튜플(tuple)이 얻어진 경우 이를 테이블(table) 형태로 표현하기 위한 기능 등이 있다. 그림 7

은 객체 표현기를 통해 웹 객체가 여러 웹 문서 형식중에서 HTML 형식으로 표현된 예를 보이고 있다(부록 B의 CodeObject와 RefObject, RelationshipTriple 사례 참조).

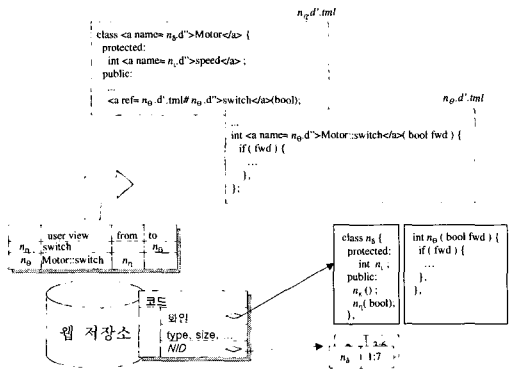

그림 7 코드 웹 객체의 표현 예

객체 분석기와 변환기를 거쳐 구축된 웹 객체는 객체 표현기를 통해 항해 관점에서 재구성된다. 세부 내역들간의 항해는 항해 식별자를 적용한 하이퍼텍스트 앵커(anchor)를 사용하여 표현하고 있다. 한편, 코드와는 달리 그래픽 형식을 갖는 웹 객체로 항해 식별자가 n_i인 객체도(n_i.d"=ε)는 그래픽 영역에서 관련 내역으로 항해할 수 있도록 하기 위해 그림 8과 같이 이미지 맵(n_i.d'.map 파일)을 함께 생성한다.

- n_i.d'.tml
   <img src=n_i.d'.gif border=0 width=394 height=273 ismap>
  ...
- n_i.d'.map
  # Map for 'Moviebox' Object Diagram
  rect n_j.d'.tml#n_j.d" 204,152 379,183
  poly n_k.d'.tml#n_k.d" 181,49 193,68 181,88 189,68 ...

그림 8 그래픽 설계내역의 표현

이미지 맵에서는 객체도에서 구현 코드로의 참조를 위한 정보를 포함한다. 이때, 객체 표현기는 X 좌표계를 웹의 좌표계로 변환 및 조정하고, 참조 가능한 다른 내역으로의 항해 정보를 위한 맵 요소들을 갖도록 생성한다.

그림 9는 일반적인 분석 과정으로 지금까지 설명한 웹 객체들을 실제로 웹 브라우져상에 보이고 항해하는 과정을 보인 것이다. 웹 객체인 'MovieBox' 객체도와 동적 관점의 동적도, 그리고 이에 대해 생성된 C++ 내역들이 객체 표현기를 통해 웹 브라우져상에 보여지는 과정을 보이고 있다.

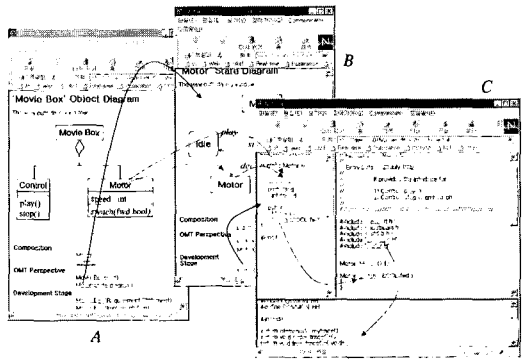

그림 9 사용자 인터페이스

표현기는 저장소내의 여러 파일과 항해 정보를 조합한 HTML 문서 뿐만 아니라 [23]을 통해 얻을 수 있는 관계성에 기반한 다른 설계 내역으로의 항해를 위한 하이퍼텍스트를 함께 제공한다. 즉, 세부적인 참조내역으로의 항해 뿐만 아니라 설계정보들간에 갖는 연관성에 따라 관련 설계 내역으로의 항해 방법을 같이 제공하고 있다. 하지만, 동적도내의 상태(state)는 특정 제어 흐름에 대한 추상적인 뷰를 나타내는 것이기 때문에, 항해의 대상을 명확히 구분할 수 없다. 따라서, 본 연구에서는 상태로의/에서의 항해를 지원하지 않고, 상태의 변화를 발생시키는 이벤트/메시지에 대한 항해만을 제공한다. 그림에서는 'MovieBox' 클래스(A 윈도우)를 보며, 세분화 관계가 있는 두 개의 클래스('Control'과 'Motor')가 존재함을 알 수 있고, 다시 이들에 대한 OMT 관점에서 'Motor'의 동적 관점을 보이는 동적도(B 윈도우)가 있음을 알 수 있다. 또한, 개발 단계 관점의 'Motor' 구현 코드(C 윈도우)를 함께 보이고 있다.

우리의 도구는 사용자가 입력한 키워드와 일치하는 설계 정보를 추출하는 검색 도구와는 달리, 한 설계정보의 내용을 쉽게 파악하기 위한 분석 인터페이스와 관련 설계정보로의 항해를 제공하여 설계정보의 이해를 돕는다는 점에서 재사용을 지원하는 분석 도구라고 할 수 있다.

5. 결론 및 향후 연구과제

본 논문에서는 사용상의 용이성과 플랫폼 독립적인 특성을 가진 웹 위에서, 호스트에 구축된 객체지향 설계 내역과 원시 코드를 웹브라우저를 통해 참조하고, 분석하기 위한 환경에 대해 기술하였다.

우리의 저장소는 이미 개발 도구 수준에서 파악한 각종 설계 의미(design semantics)들을 분석 작업에 최대한 활용하기 위해, 기존에 고안된 설계정보 저장구조[22]와 데이타 모델[23]을 연동하는 방식으로 구성하였고, 여기에 웹 환경과 항해를 위한 부분을 추가로 확장하였다. 이를 통해, 소프트웨어 개발중에 얻은 메타 정보들을 통해 내역들간의 자유로운 항해를 제공하여 소프트웨어의 효과적인 분석과 이해도 향상에 도움을 줄 수 있다. 또한, 웹을 기반으로 고안하였기 때문에, 웹 브라우져를 가진 어떤 시스템에서도 쉽게 접근할 수 있다는 특징을 갖는다.

향후 연구 과제로는 프로그램 코드의 시험 및 검증을 위한 도구들과의 통합에 대한 고려가 추가로 필요하다. 구축된 소프트웨어를 시험하기 위해 작성된 시험 프로그램과 데이타들을 함께 구조화하고, 이들에 대한 적절한 접근 수단은 분석 도구의 기능을 향상시키는 주요한 요소 중의 하나이다. 또한 저장소 내에는 현재 활용하지 않는 정제된 정보들이 존재한다. 따라서, 이 정보들을 활용한다면 보다 유용한 분석 환경을 구축할 수 있을 것이다. 한 예로, 그림 9의 설계 내역을 분석할 때, "'Control' 클래스의 메소드가 기술된 분석 단계의 모든 내역과 그에 대응하는 구현 코드는 ?"과 같은 질의로 얻은 결과는 'Control' 클래스의 분석에 매우 유용한 정보가 될 것이다.

참 고 문 헌

[1] J. A. Begoray, "An introduction to Hypermedia Issues, System and Application Areas," Int'l J. of Man-Machine Studies, Vol. 33, pp. 34-45, 1990.

[2] T. Berners-Lee and D. Connolly, Hypertext mark-up Language Specification 2.0, Internet RFC1866, Nov. 1995.

[3] T. Berners-Lee, Hypertext Transfer Protocol-HTTP/1.0, Internet RFC 1945, May 1996.

[4] R. H. Bourdcau and Betty H. C. Cheng, "A Formal Semantics for Object Model Diagrams," IEEE Trans. on Software Engineering, Oct. 1995.

[5] CCITT, ITU Specification and Description Language, Recommendation Z.100 Blue Book, Nov. 1988.

[6] Serena Coetzee and Judith Bishop, "New way to query GISs on the Web," IEEE Software, Volume 15, Issue 3, pp. 31-40, May-June 1998,

[7] Cscope, http://www.lucent.com/ssg/html/cscope.html, Lucent Technologies.

[8] B. P. Douglass, Real-Time UML: Developing Efficient Objects for Embedded Systems, Addison-Wesley, pp. 365, Dec. 1997.

[9] D. Eichmann, T. McGregor, and D. Danley, "Integrating Structured Database into the Web: The More System," Int'l Conf. on WWW, Geneva, May 1994.

[10] Scott Henninger, "Supporting the Construction and Evolution of Component Repositories," The 18th International Conference on Software Engineering, Berlin, Germany, March 1996.

[11] Hyperbook Browser User Guide Ver 2.3, Computer Command and Control Company, June 1998.

[12] INSYDE, "Integrated Methods for Evolving System Design," ESPRIT-III Project P8641, restricted report edition, Dec., 1994.

[13] S. P. Hadjiefthymiades and D. I. Martakos, "Improving the Performance of CGI Compliant Database Gateway," Computer Network and ISDN Systems, Vol. 29, pp. 1291-1304, 1997.

[14] Z. Navavi, VHDL Analysis and Modeling of Digital Systems, McGraw-Hill, 1993.

[15] pcGRASP v6.2.8, http://www.eng.auburn.edu/department/cse/research/grasp/, Auburn Univ., 1998.

[16] D. Robinson, The WWW Common Gateway Interface version 1.1, Internet draft, Jan. 1996.

[17] J. Rumbaugh et al., Object-Oriented Modeling and Design, Prentice Hall, 1991.

[18] G. J. Yaverbaum and J. Liebowitz, "GoFigure INC: A Hypermedia Web-based CASE," J. of Computer Educ., Vol. 30, No. 3-4, pp. 147-156, 1998.

[19] E. Y. Wang, H. A. Richter, and Betty H. C. Cheng, "Formalizing and Integrating the Dynamic Model within OMT*," Proc. of IEEE Int'l Conf. on Software Engineering, May 1997.

[20] B. B. Welch, Practical Programming In Tcl & Tk, 2nd Edition, Prentice-Hall, pp. 630, 1997.

[21] Argo/UML v0.7: The Cognitive CASE Tool, http://argouml.tigris.org/, Regents of the Univ. of California, 1998.

[22] 배명남, 김훈희, 양재동, 최완, "객체 모델링을 지원하는 의미 기반 설계 도구의 개발", 한국정보과학회 논문지(C), Vol. 3, No. 3, pp. 248-261, 1997.

[23] 양재동, 배명남, 장재우, 이준경, "객체 지향 개발 환경을 지원하는 항해형 데이타 모델", 한국정보과학회 논문지(B), 제 25권 1호, pp. 85-98, 1998.

부록 A. 웹 객체의 스키마

```
create class ObjectID (
);
create class DesignObjectID as subclass of ObjectID (
);
create class ReferenceObjectID (
);

create class NavigationIdentifier (
design_object_id_DesignObjectID,
reference_object_id_ReferenceObjectID
);
create class RelationshipTriple (
source_object_NavigationIdentifier,
relationship_kind_RelationshipKind,
target_object_NavigationIdentifier,
user_view_string
);
create class WebObject (
original_design_object_id_DesignObjectID
);
create class RefObjectID (
source_location_FigureInfo,
source_nid_NavigationIdentifier
);
create class ClassDiagramObject as subclass of WebObject (
 diagram_file_DiagramFileObject,
ref_objects_id_set(RefObjectID)
);
```

부록 B. 그림 5,6,7에서 소개되고 있는 예의 스키마 사례들

NavigationIdentifier

	*design_object_id_*	*reference_object_id_*
n_η	<motor.cc,분석,C,1.0>	<1>
n_θ	<motor.cc,분석,C,1.0>	<2>
n_δ	<motor.cc,분석,C,1.0>	<3>
n_κ	<motor.cc,분석,C,1.0>	<4>
n_ι	<motor.cc,분석,C,1.0>	<5>
n_β	<motor.om,분석,CD,1.0>	<1>

RelationshipTriple

	*source_object_*	*relationship_kind_*	*target_object_*	*user_view_*
	n_η	진화	n_θ	switch
	n_θ	정의	n_η	Motor::switch
	n_β	객체정의	n_δ	Motor

ClassDiagramObject

	*diagram_file_*	*ref_object_id_*
	motor.om	{ 5 }

CodeObject

	*code_file_*	*ref_object_id_*
	motor.cc	{ 1,2,3,4 }

RefObjectID

	*source_location_*	*source_nid_*
1	8:5	n_θ
2	1:7	n_δ
3	5:3	n_κ
4	3:6	n_ι
5	462,353,393,80	n_β

배 명 남

1991년 2월 전북대학교 전산통계학과(학사). 1993년 2월 전북대학교 전산통계학과(석사). 1998년 2월 전북대학교 전산통계학과(박사). 1998년 3월 ~ 현재 한국전자통신연구원 실시간 DBSM팀(선임연구원). 관심분야는 분산객체, 실시간 시스템, CASE.

최 완

1981년 경북대학교 전자공학과(전자계산학 전공, 학사). 1983년 한국과학기술원 전산학과(석사). 1983년 ~ 1985년 한국과학기술원 전산학과 연구조교. 1988년 정보처리 기술사(전자계산응용) 자격소지. 1985년 ~ 현재 한국전자통신연구원 실시간 DBMS팀장. 관심분야는 CASE, Compiler, OS, DBMS.

양 현 택

1985년 전남대학교 전산통계학과(학사). 1998년 순천대학교 컴퓨터과학과(석사). 1999년 ~ 현재 순천대학교 컴퓨터과학과 박사과정. 관심분야는 소프트웨어 공학, 객체지향개발방법론.