객체지향 모형 간 일관성 검증을 지원하는 CASE 도구 설계 및 구현

이 선 미[†] · 전 진 옥[††] · 류 재 철[†††]

요 약

개발하고자 하는 소프트웨어 시스템을 표현하는데 다양한 모형과 그에 따른 다이어그램들이 사용된다. 그러나 이들은 궁극적으로 하나의 시스템으로 통합되고 구현되어져야 한다. 그러므로 소프트웨어 개발 과정에서 생성되는 모형 정보들은 일관성을 유지하고 있어야 한다. 객체지향 도구를 이용하여 소프트웨어 시스템을 모델링하는 개발자들이 모형간 논리적 오류를 포함하지 않은 모형을 개발할 수 있도록 하기 위하여, 본 논문에서는 소프트웨어 아키텍처에 대한 메타 모형을 제시하고 이 메타 모형을 통한 일관성 규칙을 제시한다. 또한 이 규칙을 객체지향 CASE 도구인 DEBUT에 적용하여 논리적 오류 자동 검출 기능 구현 및 그에 따른 기대효과를 제시한다.

Design and Implementation of a CASE Tool Supporting Proof of Consistency between OO Models

Sunmi Lee[†] · Jin-Ok Jeon[††] · Jae-Chol Ryou[†††]

ABSTRACT

There are several models and the corresponding diagrams to express software system in many kinds of viewpoints, but these are supposed to be integrated and implemented into only one system. Therefore, the software modelers should have the models ensuring the consistency between information in software development life cycle. To support the robust models for modelers using OO modeling methods, i.e. UML, and CASE tools, the meta models of the software architecture and the consistency rules between the models are suggested in this thesis. Finally, the rules are implemented in the OO CASE tool, DEBUT(Design By UML Tool). It supports UML 1.1 notations and has visual modeling editors that enable users make their own software model.

1. 서 론

소프트웨어 시스템의 규모가 크고, 기능이 복잡할수록, 이의 개발을 책임지고 있는 조직에서는 프로젝트 팀을 구성하고, 개발 대상 시스템의 특성에 맞는 적절한 방법론과 개발 도구들을 사용한다. 시스템 개발방법론은 소프트웨어 생성을 위한 개발 단계를 정의하고, 활동, 산출물, 검증 절차, 각 단계의 완결 조건을 명시하는 체계적인 방법[1] 등을 정의하고 있다. 그러므로 정보기술의 발전에 따른 신기술의 계속적인 수용과 사용자 요구사항 변화의 효율적인 수용, 시스템 개발의 생산성 제고 및 품질의 보증을 위해서는 개발방법론과 개발도구의 활용이 필수적이다.

90년 이후 최근까지 정보시스템 개발을 위해 객체지향 기술에 기반한 시스템 개발이 채택되고 있다. 객체

† 정 회 원 : 전자통신연구원 실시간 컴퓨팅연구부 선임연구원
†† 종신회원 : 전자통신연구원 책임연구원 실시간컴퓨팅연구부장
††† 종신회원 : 충남대학교 컴퓨터과학과 교수
논문접수 : 1999년 7월 1일, 심사완료 : 1999년 10월 28일

지향 기술은 복잡하고 분산화된 시스템을 개발하고 관리하는데 적합한 기본적인 패러다임으로 인식되고 있으며, 객체지향 프로그래밍 언어, 분산 객체 기술, 객체지향 데이터베이스의 발전과 급속한 보급은 객체지향에 기반한 정보시스템의 개발을 가속화하고 있다. 객체지향에 기반한 정보시스템의 개발은 단순한 개발도구의 변경을 의미하는 것이 아니라 개발될 시스템을 바라보는 기본적인 개념, 패러다임의 변화를 의미한다.

80년대 말부터 객체지향 개발방법론에 대한 연구가 활발히 진행되어 90년대 초에는 OOD(Object-Oriented Design), OMT(Object Modeling Technique), OOSE(Object-Oriented Software Engineering) 등 20여 개의 개발 방법론들이 제시[2, 3, 4]되었으며, 다수의 객체지향 개발방법론의 출현은 객체지향 개발방법론의 보급을 저해하고, 도구간의 상호 작용성(interoperability), 모형화 산출물간의 변환 등의 문제를 야기하였다. 이에 따라 90년대 중순부터 표준화된 객체지향방법론에 대한 논의가 활발히 대두되었다. 그 결과로 1997년 가을에 OMG (Object Management Group)에서 UML(Unified Modeling Language)을 표준화된 객체지향 모형화 언어로 채택하게 되었다. 이로 인해서 객체지향 개발방법론 및 UML의 표기법을 지원하는 다수의 CASE 도구들이 제작되고 있으며 향후 방법론 및 관련 도구의 실제적 적용은 급격히 늘어날 것으로 예상된다[5].

그러나 UML은 용어 그대로 단지 모형을 정의하는데 필요한 표기법과 각 모형요소에 대한 의미만을 개발자에게 제공하는 모형화를 위한 통합 언어(Unified Modeling Language)이므로, 각 모형들이 언제(When), 어떤 방식으로(How) 사용되어야 하는지에 관한 가이드가 요구되었다. 이에 UML의 제정을 적극적으로 주도해 온 Rational사는 1998년에 UML의 보다 효과적인 사용을 목적으로 UML을 활용하는 소프트웨어 개발 과정과 개발 단계를 정의한 통합 프로세스(Unified Process, 이하 Unified Process라 표기한다.)[6]를 개발하였다.

Unified Process에 따르면, 프로세스는 누가 무슨 일을 하는지(Who is doing What), 그 일은 언제 해야 하는 지(When to do it), 어떻게 주어진 목표를 달성하는지(How to reach a certain goal)를 기술한 것으로써, 2가지 측면에서 설명된다. 첫째는 시간의 흐름에 따른 개발조직의 시스템 개발 과정(Phase)을 정의하는 관점이며, 두번째는 시간의 흐름을 배제한 개발 조직이 수행해야 하는 작업(Workflow)의 관점이다.

Unified Process의 특징은 첫째, 기본적으로 반복적이며, 점진적이다. 즉 시스템 개발을 위한 모든 단계(Lifecycle Phases)에서 특정 임무 수행을 위한 활동(Activity)을 반복적이며 점진적으로 수행한다. 둘째, 요구사항 분석, 설계, 구현 및 시험 활동 전반에 걸쳐 사용사례(Use Case)가 기반이 된다는 점이다. 셋째, Unified Process에 의하면 개발될 소프트웨어 시스템의 모형들은 아키텍처(Architecture)를 정의하고, 가시화하고 구축하고, 문서화하기 위한 수단이다.

UML과 Unified Process의 적극적인 보급과 함께 소프트웨어 개발에서 견고한 시스템 아키텍처를 정립하는 것이 프로젝트 성패에 중요한 요인으로 인식되고 있다. 개발 초기에 확고한 아키텍처를 정립하면 재작업을 줄이고, 재사용성, 확장성, 시스템 품질 측면에서 많은 장점을 얻을 수 있다[7]. 시스템 아키텍처는 (1) 시스템 구성요소들의 구조, (2) 구성요소간의 상호관련성, (3) 설계와 시간에 따른 시스템 진화를 관리하기 위한 원칙과 지침으로 정의할 수 있다. 소프트웨어 시스템의 아키텍처는 인터페이스를 통해 상호 작용하는 시스템의 구성요소(component)의 조직이나 구조를 말하며 시스템의 제약사항(constraints)과 시스템 구성요소의 구조에 대한 근거(rationale)를 포함한다[7]. 소프트웨어 아키텍처 정립의 중요성은 개발관련자간 상호 의사소통의 지원, 초기의 설계 의사결정의 명문화, 시스템 이전의 용이성 등에 있다. 아키텍처의 정립은 시스템의 병행적인 개발, 재작업의 최소화, 재사용성 및 유지보수성의 향상 등을 가능하게 한다.

소프트웨어 아키텍처를 표현하기 위해서는 여러 가지 방안이 활용될 수 있으나, 하나의 다이어그램으로는 소프트웨어 아키텍처를 전체적으로 표현하는 데는 한계를 갖는다. 따라서 일반적으로 여러 관점에서 소프트웨어 아키텍처를 표현하고 검토하는데, UML이 표준으로 대두되기 이전의 대표적인 표현 방법은 소프트웨어를 구조적인 관점에서 파악하는 객체 모형 (class model), 소프트웨어의 기능을 표현하는 관점인 기능 모형(functional model), 소프트웨어의 동적인 상태를 표현하는 동적 모형(dynamic model) 등 세가지 관점에서 표현하였다. UML이 제정된 이후 대표적으로 활용되는 아키텍처 표현 방법이 사용사례를 중심(Use Case Driven)으로 한 4+1 관점(view)[8]이다. 이에 대해서는 3장에서 상세히 소개한다.

CASE 도구를 사용하여 시스템을 개발할 경우 다이어그램 편집기를 통하여 시스템 분석가 및 개발자의 모형화 작업에 의해 생성된 모형결과는 프로젝트 전체 진행 과정에서 주요한 산출물 역할을 함과 동시에 CASE 도구 기능의 일부로 지원되는 원시코드 자동 생성의 입력물로 제공된다. 즉 보다 완벽한 모형은 견고한 시스템 구현을 위해 필수라 하겠다.

본 논문의 구성은 다음과 같다. 2장에서는 정보 시스템 개발 시 시스템 분석가 및 설계가들의 모형화 작업을 지원하는 CASE 도구의 기능 및 역할과, CASE 도구에서 모형간 일관성을 지원하기 위한 연구사례들을 살펴본다. 3장에서는 시스템 아키텍처와, UML의 표기법을 이용하여 이 아키텍처를 표현하는 여러 모형들의 특징과, 이 모형들을 구성하고 있는 모형요소들은 어떤 것들이 있는지, 어떤 상호 연관관계를 맺고 있는지 UML 표준 메타 모형을 근간으로 분석한다. 4장에서는 객체지향 CASE 도구 내에서 사용자의 모형 결과를 분석하는 모형 검증 절차 및 메타 모형을 정의하고, 정의된 메타모형으로부터 검증 규칙 및 유형을 추출하고 정의하였으며, 5장에서는 검증 규칙의 CASE 도구 적용 사례를 제시한다. 6장에서는 결론으로 기대 효과 및 향후 연구방향에 대하여 언급한다.

2. 관련 연구

2.1 CASE 도구와 모형 검증 기능

소프트웨어 개발 방법론의 보급과 함께 소프트웨어 개발 자동화를 위하여 도입하게 되는 CASE 도구는 프로젝트 관리 도구, 시스템 분석 설계 도구, 사용자 인터페이스 설계 도구, 프로그래밍 개발 환경에 이르기까지 매우 다양하다. 이전에는 각 도구들이 개별도구로 존재하여 시스템 개발 단계마다 각각의 도구를 활용하였다. 따라서 도구간 정보의 호환성 및 일관된 표현에 어려움이 있었다. 그러나 최근에는 분석 설계를 지원하는 모형화 도구를 중심으로 형상관리 도구, 프로그램 개발 도구 또는 테스트 자동 지원 도구 등이 통합화 추세에 있어 도구간 정보의 호환 및 연계를 지원하고 있다.

이중 정보 시스템 개발에 가장 핵심이 되는 도구는 실세계로부터 문제점을 추출하여 이를 추상화된 표기법으로 표현하는 모형화 과정을 지원하는 시스템 분석 및 설계 도구이다. 모형의 기본 요소인 표기법(notation) 및 그 의미(semantics)는 방법론마다 상이하므로 모형화를 지원하는 분석 설계 도구는 특정 방법론에 매우 의존적이나, 방법론과 무관하게 CASE 도구가 사용자들에게 제공하는 기능은 다음과 같다.

- 그래픽 모형 편집 기능 : 모형을 구성하고 있는 모형요소들을 아이콘 및 기타 그래픽 사용자 인터페이스를 이용하여 생성 및 삭제하고, 값의 변동 등을 대화식으로 화면을 통하여 확인 가능한 모형화 작업 지원 기능
- 정보 관리 기능 : 사용자가 작업한 모형의 저장 및 복원이 가능하도록 하며, 모형정보의 계층화된 정보관리 지원
- 문서 생성 기능 : 그래픽 편집기를 통하여 작업한 모형들에 대한 정보를 다양한 형태의 문서로 생성.
- 원시코드 생성 기능 : 소프트웨어 시스템의 모형의 각 모형요소에 대한 명세를 이용하여 컴파일 가능한 프로그램 코드로 생성.
- 규칙 베이스에 의한 사용자 모형화 작업 지원 기능 : 모형의 주된 요소인 개체와 개체를 연결시켜주는 선들에 대한 연결 규칙과, 모형의 타당성 점검을 통하여 정확한 모형화 작업 지원.
- 타 도구간 호환성 지원 : 사용자에 의해 생성된 모형으로부터 테스트 케이스를 자동추출하는 테스팅 지원도구, 자동생성된 원시코드를 위한 프로그래밍 환경, 모형 또는 원시코드에 대한 형상 관리 도구 등 타 도구와의 연계성

CASE 도구의 가장 중요한 기능은 물론 사용자의 편의성을 극대화 한 모형 편집 기능이지만, 생성된 모형이 소프트웨어 개발 라이프 사이클 내 각 단계에서 연계되기 위해서는 모형에 대한 정확한 검증 기능은 반드시 제공되어야 한다.

2.2 개발과정과 CASE 도구의 역할

소프트웨어 개발 단계를 작업 단위로 나누면 계획, 분석, 설계, 구현, 테스트, 및 유지보수로 분류하고 이를 소프트웨어 생명주기(Software Development Life Cycle)라고 부른다. CASE 도구는 이 생명주기 단계별 활동별로 다음과 같이 사용될 수 있다.

- 분석 : 요구사항 분석에 의해 추출된 객체 및 객체간 관계의 시각적 표현 및 사용자 참여 유도

- 설계 : 개발 대상 시스템의 구조 및 기능의 구체화 및 상세화, 시스템 개발 시 객체관리의 자동화, 시스템 재공학 시 기존 코드로부터 객체 모델 복원
- 프로그래밍 : 분석 및 설계 모델로부터 자동 생성된 코드의 활용, 설계 변경 시 일관성을 유지하면서 코드의 변경
- 테스팅 : 모델링 결과로부터 자동 추출된 테스트 케이스 활용
- 프로젝트 관리 : 대규모 프로젝트 수행 시 모델 및 코드의 형상 관리, 개발과정에서 요구되는 문서화 작업의 자동화

위에서 보는 바와 같이 CASE 도구는 소프트웨어 생명 주기 전 과정을 통하여 사용되며 그 모형화 작업 결과 역시 다양한 형태로 반영되어 견고한 정보시스템 개발의 기반이 된다. 소프트웨어 모형은 개발 초기에는 사용자 또는 시스템 개발의 발주자와의 정확한 시스템 도출을 위한 대화 시 매우 중요한 커뮤니케이션 수단으로 사용된다. 그리고 대형 소프트웨어 시스템 개발 시에는 시스템을 소규모로 분할하여 모형화 작업이 이루어지게 되며, 부분적으로 완성된 모형들이 전체 시스템 내에서 조화를 이루기 위해서는 모형 단계에서 통합된 모습을 갖추고 있어야 한다. 이때에도 모형은 개발자들 간 중요한 커뮤니케이션 수단으로 사용된다. CASE 도구를 통하여 모형이 개발되는 경우, 모형을 구성하는 텍스트 명세는 CASE 도구에서 제공하는 원시코드 생성 기능에 의하여 프로그램 코드로 자동 변환되므로 전체 모형의 정확성 및 일관성은 더욱 중요한 과제라 아니할 수 없다.

2.3 일관성 검증 연구 사례

UML이 OMG에 의해 객체지향 모형의 표준으로 제정되기 이전에도 소프트웨어의 표현 방법인 여러 모형에 대한 일관성 검증에 관한 연구가 있어왔지만 그 연구 범위가 매우 미미하다. 기존의 구조적 방법론을 연구하던 많은 방법론 연구가들이 연구 대상을 객체지향 방법론으로 옮겨가고 있었으며, 방법론의 정립, 즉 모형화 과정이나 모형의 표현 방식 등에 많은 시간과 노력을 할애하고 있었기 때문에 모형 간 일관성 검증에 대한 연구가 이루어질 수 없었다.

그러나 소프트웨어를 추상화하여 표현하는 모형들이 모형마다 상이한 표기법과 상이한 관점을 표현하고 있지만, 이 모형들이 하나의 시스템 내에서 실현되어야 하기 때문에 모형 간 일관성 검증에 대한 연구 또한 객체지향 방법론 및 표기법의 정립과 병행하여 연구가 필요하다.

90년대 중반 객체지향 방법론을 지원하는 CASE 도구가 출현하면서부터 이에 대한 연구도 조금씩 이루어지고 있다. 그러나 UML 이전의 객체모형과 상태모형을 대상으로 모형 간 상호 참조표를 통한 일관성 검증 연구[9] 또는 UML 다이어그램의 모형 요소 중 객체를 위주로 매트릭스를 작성하고, 이 매트릭트를 이용한 일관성 규칙 추출[10] 등과 같은 방법은 제한적이고, 체계적이지 않아 복잡하고 다양한 UML의 모든 모형 간의 일관성을 포괄하여 다루기에는 어려움이 있다. 다이어그램을 구성하고 있는 모형요소 중 객체가 시스템의 구성요소를 나타내는 역할을 하나, 개체와 객체를 연결하는 관계(Relationship) 역시 시스템 구성 요소들간의 유기적인 관계를 나타내고 있기 때문에 이 관계 정보 역시 모형간 일관성 검증에 중요한 요인이 된다.

신규로 정보시스템을 구축하고자 계획하거나, 기존의 시스템을 재공학을 통하여 새로운 기능을 추가하기를 원하는 조직에서는 객체지향 방법론을 적극 수용하며, 방법론을 지원하는 고가의 CASE를 구입한다. 그러므로 사용이 편리하고 쉬운 그래픽 기능을 갖춘 모형 편집기능과 보다 완벽한 모형 검증 기능을 갖춘 CASE 도구의 개발이 시급하다 할 수 있겠다.

3. S/W 아키텍처별 UML 메타 모형 제안

3.1 S/W 아키텍처

소프트웨어 시스템의 규모와 복잡도가 증가함에 따라 전체 시스템 구조를 설계하고 명세화 하는 것이 이를 컴퓨터에 구현하기 위하여 알고리즘이나 자료구조를 선정하는 것보다 더욱 중요한 일이 되었다[11]. 제1장에서 언급한 바와 같이 소프트웨어 아키텍처는 구성요소들 간의 구조, 구성요소들 간의 상호 관련성 및 설계 및 시간의 경과에 따른 시스템 진화를 관리하기 위한 원칙과 지침으로 정의된다. 구조는 시스템을 구성하고 있는 구성요소들이 어떻게 조직화되고 있는지, 물리적으로 어떻게 분산될 것인지, 구성요소들 간 기능을 어떻게 분산할 것인지 등을 포함하고 있다. 아키텍처는 이러한 구조적인 측면들만을 표현하는 것이 아

니라, 이 구성요소들이 상호 어떤 작용을 하며 어떤 정보들을 주고 받는지에 관해서도 구현 단계가 아닌 분석 설계 단계에서 정의된다.

위의 정의로부터 볼 때 소프트웨어 아키텍처는 하나의 시스템의 어느 특정한 국면을 특정한 목적으로 추상화, 단순화하여 형상화하는 것이기 때문에 다루고자 하는 관점과 관련이 없는 부분은 해당 아키텍처로부터 제거시키게 된다[12]. 그와 같은 연유로 단일 모형 또는 단일 다이어그램을 이용하여 시스템 아키텍처를 모두 표현할 수는 없다.

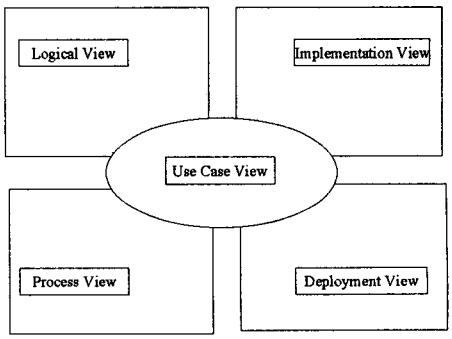
소프트웨어 아키텍처를 표현하기 위하여 여러가지 방안이 활용된다. Rumbaugh의 OMT(Object Modelling Technique)에서 소프트웨어를 객체 모형(Object Model), 기능 모형(Functional Model), 동적 모형(Dynamic Model)로 나누어 모형화 하도록 제안[4]하였으며, G. Booch는 아키텍처를 정적 모형(Static Model), 논리 모형(Logical Model), 동적 모형(Dynamic Model) 및 구현 관점에서의 물리적 모형(Physical Model)로 구분하였다[2]. 한편 I. Jacobson은 요구사항 모형(Requirement Model), 분석 모형(Analysis Model), 설계 모형(Design Model), 구현 모형(Implementation Model), 시험 모형(Test Model)으로 구분[3]하여 소프트웨어 생명주기에 따른 다양한 모형 및 그에 따른 다이어그램들을 제시하였다.

앞에 언급한 세 사람이 후에 각자 자신의 방법론 및 모형과 여타 객체지향 방법론 등을 통합하여 제시한 통합 객체지향 모형이 UML(Unified Modeling Language)이다. 나아가 UML과 더불어 소프트웨어 아키텍처를 표현하는 방식으로 4+1 관점(View)을 제시하고 있다.

3.2 4+1 관점(View) 과 UML 다이어그램

다음 (그림 1)에서 보는 바와 같이 4+1 관점은 소프트웨어 시스템을 5가지 관점으로 보고 있다. 5가지 관점은 사용사례 관점(Use Case view)을 중심으로, 논리 관점(logical view), 프로세스 관점(process view), 구현 관점(implementation view), 배포 관점(deployment view)으로 구성되고 있으며, 논리 관점과 프로세스 관점은 개념적(Conceptual) 관점에서의 아키텍처이며, 구현 관점과 배포 관점은 물리적(Physical) 관점에서의 아키텍처이다.

다음은 각 관점에 대한 간단한 설명이다.



(그림 1) 4+1 관점

- 사용사례 관점 : 최종 사용자의 관점이며, 시스템이 사용자에게 제공하게 될 기능이 주된 관심사이다.
- 프로세스 관점 : 시스템 통합을 담당하는 개발자의 관점이며, 데이터 처리량, 응답 시간 등 시스템의 성능이 주된 관심사이다.
- 구현 관점 : 프로그래머의 관점으로서, 소프트웨어를 구현하고 관리하는데 초점을 둔다.
- 배포 관점 : 개발 완료 후 시스템 인도, 설치 등 시스템 엔지니어링 관점이다.

UML은 소프트웨어 아키텍처를 다양한 관점에서 모형화 할 수 있도록 클래스도(Class Diagram), 사용사례도(Use Case Diagram), 순서도(Sequence Diagram) 등 9가지 다이어그램을 위한 모형요소(Model Element)들을 정의하고 있다. 이중 주로 업무 프로세스 모형화에 사용되는 활동도(Activity Diagram)를 제외한 각 다이어그램을 통하여 개발자들은 다음과 같은 모형화 작업을 할 수 있다.

- 사용사례도 : 사용자에게 보여지는 시스템의 기능 표현
- 클래스도 : 시스템에 존재하게 될 클래스(Class) 정의 및 클래스 간 관계 표현
- 객체도(Object Diagram) : 클래스도와 같으나 모형화의 대상이 인스턴스인 객체(Object)간 관계 표현(객체도는 협력도와 모형요소가 동일하므로 추후 연구대상에서 제외함)
- 순서도 : 시스템의 동적인 행태(제어의 흐름)를 시간 순서에 의거하여 표현
- 협력도(Collaboration Diagram) : 시스템의 동적인 행태를 표현하는데, 객체(Object)의 구조 및 메시

지 흐름을 위주로 표현

- 상태도(Statechart Diagram) : 이벤트에 반응하는 클래스의 상태 표현
- 구성요소도(Component Diagram) : 구현과 관련된 실제적인 시스템의 구조를 표현
- 전개도(Deployment Diagram) : 배포될 시스템의 하드웨어적 형상 표현

소프트웨어 아키텍처를 위의 4+1 관점으로 분류하고, 이 관점에 상응하는 소프트웨어 모형화 작업을 지원하는UML의 다이어그램을 <표 1>과 같이 분류할 수 있다.

〈표 1〉 S/W 아키텍처와 다이어그램 분류표

아키텍처	UML 다이어그램
사용사례 관점	사용사례도, 순서도,
논 리 관 점	클래스도, 순서도, 협력도, 상태도
프로세스 관점	협력도
구 현 관 점	구성요소도
배 포 관 점	전개도

본 논문의 주요 연구분야는 UML의 모형 간 일관성 검증이므로, 소프트웨어 아키텍처로 4+1 관점을 따르며, 다음 절에서 그에 따른 소프트웨어 아키텍처와 UML 다이어그램 간 관계를 메타 모형화 하고, UML의 각 다이어그램을 각 아키텍처 관점에 따라 메타 모형화 하였다.

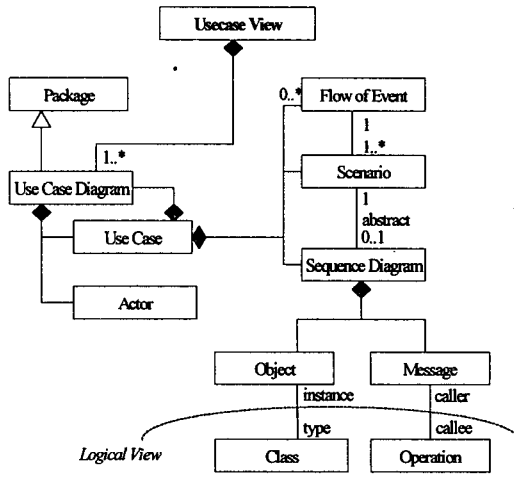
3.3 메타 모형

3.3.1 사용사례 관점과 메타 모형

사용사례도를 구성하고 있는 모형요소는 사용사례(Use Case), 행위자(Actor), 그리고 사용사례와 사용사례간 일반화(Generalization) 관계와 행위자와 사용사례간 연관(Association)관계로 구성된다.

UML 표기법 안내서[5]에 의하면 사용사례는 연산(Operation)과 속성(Attribute)을 포함하고 있는 classifier의 부분집합이다. 즉 사용사례에서의 연산이란 해당 사용사례 내에서 수행하게 될 기능을 말하므로, 이 기능은 곧 순서도에 나타나는 객체와 객체가 주고 받는 메시지와 연관을 맺게 된다.

각 사용사례는 시스템이 수행하게 될 기능을 나타내므로 기능을 수행하는 순서를 모형화하는 순서도와 연관관계를 맺는다. 즉 순서도는 사용사례도의 직접적인 모형요소는 아니나, 사용사례도 내의 한 모형요소와 간접적인 관계를 맺고 있는 것이다. 그러므로 메타 모형에서는 이것이 반영된다.



(그림 2) 사용사례 관점 메타 모형

3.3.2 논리적 관점과 메타 모형

시스템을 논리적 관점에서 표현하는 데에는 UML 여러가지 다이어그램이 활용된다. 그러므로 논리적 관점에 대한 메타 모형은 매우 신중해야 한다. 본 논문에서는 UML의 Notation Guide에 제시되어 있는 UML의 모형요소간 메타모형을 기본 모형으로 삼고 있다.

- 클래스도와 메타 데이터

클래스도는 구조를 표현할 많은 모형요소와 관계 및 제약사항 관련 메타 데이터들로 구성된다. 사각형 박스는 클래스를 나타내며 클래스는 연산과 속성을 포함하고 있다.

클래스는 인터페이스 클래스와 템플릿 클래스라고 하는 특수한 클래스로 세분화될 수 있다. 인터페이스 클래스는 Java또는 CORBA의 IDL과 같은 언어에서 사용되는 개념으로 다른 클래스들과의 메시지 교환을 위한 연산은 정의되어 있으나 그에 대한 구현은 타 클래스에 의존하는 클래스를 말한다. 이 인터페이스 클래스는 클래스도에서 정의되어 사용되며, 구성요소도와 같은 다른 모형에서도 모형요소로 나타날 수 있다. 템플릿 클래스는 그 자체로 완벽한 클래스가 아니기 때문에 다른 클래스와 어떠한 연관(Association) 관계

도 맺지 못한다. 이와 같은 제약사항도 모형 검증 시에 점검되어야 한다.

클래스와 클래스를 연결하고 있는 연결자에는 연관(Association) 관계, 일반화 관계 및 종속(Dependency) 관계, 실현(Realization) 관계 등이 있다. 연산, 속성, 연관관계 등은 각각의 성질을 결정하는 다수의 구성요소를 포함하고 있다. 다음 <표 2>는 각 모형 요소별 세부 속성이다.

〈표 2〉 모형요소 별 세부 속성

메타 데이터	세부 속성
클 래 스	이름, 타입, 스테레오 타입
연 산	이름, 매개변수, 반환 값, 가시성(Visibility)
속 성	이름, 타입, 초기값
연 관 관 계	가시성, 한정자(Qualifier), 역할명, 다중성
종 속 관 계	스테레오 타입

이외에도 소프트웨어의 논리적 관점에서는 시스템의 직접적인 구성요소가 아니지만, 다른 클래스도와 종속(Dependency) 관계를 갖거나, 구현 관점에서 구성요소도의 구성요소(Component)와 연관을 가지는 패키지도 클래스도의 모형요소로 포함된다.

● 순서도와 메타 데이터

순서도에서 사각형으로 표현되는 모형요소는 객체(Object)를 의미하며 횡으로 연결되어 있는 화살선은 객체들이 주고받는 메시지를 의미한다. 메시지는 메시지를 보내는 객체와 메시지를 받는 객체의 사이에서 정의되는데 메시지를 받는 객체(Receiver Object)는 관련 메시지를 수행할 연산을 보유하고 있어야 한다.

수직으로 이어지는 점선은 객체의 수명기간에 대한 표현(life line)이며, 그 위에 존재하는 긴 사각형은 객체가 활성화되는 기간에 대한 표현(focus of control)이다.

또한 생명선의 하단에 X 표시가 되어 있는 객체는 특정기간 동안만 일시적으로 존재하는 객체(Transient object)를 의미하므로 동일 객체가 다른 모형에서 사용되었을 경우 이 정보의 일관성 여부 또한 점검되어야 한다.

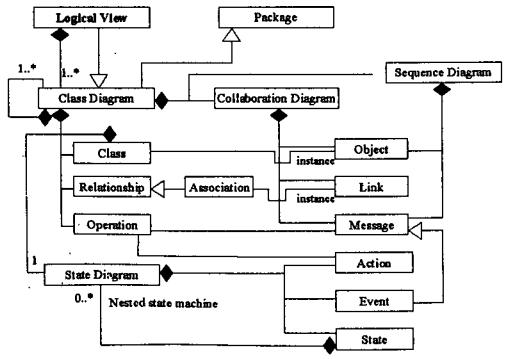
● 협력도와 메타 데이터

객체도가 클래스 간 구조 정보에 대한 인스턴스 레벨에서의 모형이라면, 협력도는 클래스도에서 표현하고 있는 클래스 간 구조 정보와 순서도에서 표현하고 있는 객체간 상호 메시지 교환 정보를 모두 가지고 있다. 그러므로 협력도는 클래스도와 일관성을 유지할 조건들을 가지고 있으며, 그 예는 다음과 같다.

- 협력도의 객체는 클래스도의 클래스들 중 한 클래스의 인스턴스이다.
- 협력도에 표현되어 있는 객체와 객체 간 링크(link)는 클래스도의 클래스와 클래스 간 연관(Association)의 인스턴스이다.
- 협력도에서 객체와 객체 사이에 표현되어 있는 메시지는 객체를 대표하는 클래스의 연산 중 하나이거나, 상태도에의 클래스가 받는 이벤트일 수 있다.
- 협력도에 표현되는 활성 객체(Active object)는 프로세스 관점에서는 컴포넌트(또는 Thread)로 표현된다.

● 상태도와 메타 데이터

상태도는 시스템의 구조적인 면 분석을 통하여 추출된 클래스들이 외부로부터 들어오는 자극에 대하여 어떻게 반응하는지를 정의하는 관점이다. 외부로부터의 자극 또한 다양하게 정의될 수 있다. 그러므로 이 상태도를 통하여 정의될 수 있는 클래스 정보는 이벤트와 행동이다. 이 행동은 해당 클래스의 연산으로 매핑된다.



(그림 3) 논리적 관점 메타 모형

위와 같이 클래스도, 순서도, 협력도 등 다이어그램을 구성하고 있는 모형요소들을 이용하여 작성한 메타 모형이 (그림 3)에 있는 논리적 관점에서의 메타 모형이다. 이 메타 모형에서는 클래스도, 순서도, 협력도

등과 같이 다이어그램도 메타 데이터로서 모형의 일부분을 구성하고 있다. 즉 순서도 협력도는 클래스도의 구성요소이며, 상태도는 클래스를 구성하고 있는 구성요소의 일부분이다. 왜냐하면 클래스도를 통하여 규정된 클래스와 클래스간 구조적인 관계는 순서도 또는 협력도를 통하여 구체화된다고 볼 수 있기 때문이다.

3.3.3 구현 관점과 메타 모형

구현 관점이란 프로그램 언어를 이용하여 소프트웨어 시스템을 직접 구현할 프로그래머와, 프로그래머에 의해 개발된 소프트웨어들을 어떻게 유지 관리할 것인가에 관심을 두고 있는 소프트웨어 매니저의 관점에서 소프트웨어 시스템을 미리 설계해 보자는 것이다. 이 관점은 앞 절의 논리적 관점과 밀접한 연관관계를 가지고 있다. 객체지향 소프트웨어 분석 설계 및 객체지향 프로그램은 추상화 레벨이 다르고 표현 형식이 다를 뿐 개념은 하나이기 때문에 분석 및 설계에서 추출된 클래스 및 객체는 곧바로 프로그램 언어로 매핑이 가능하다.

구현 관점을 모형화 하는 데에는 구성요소도(Component Diagram)가 활용되며 구성요소도는 컴포넌트와 컴포넌트 간 상호 연관관계를 이어줄 종속관계(Dependency)로 이루어 진다.

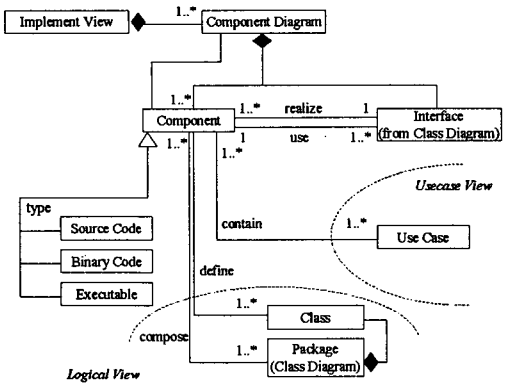
컴포넌트는 그 유형에 따라 원시코드를 의미하는 컴포넌트, 라이브러리를 의미하는 컴포넌트, 실행프로그램을 의미하는 컴포넌트로 구별할 수 있다. 각 컴포넌트들은 실제로 사용자의 컴퓨터 디스크에 설치되어질 소프트웨어들에 대한 단위로써, 논리적 관점에서 추출된 모든 클래스들은 빠짐없이 이 컴포넌트에 할당되어야 할 것이다. 즉 컴포넌트는 클래스들의 집합으로 구성된다. 그러므로 구성요소도와 클래스도는 일관된 정보를 보유하고 있어야 한다. 특히 CASE 도구는 모든 클래스들이 빠짐없이 컴포넌트에 할당될 수 있도록 사용자 편의를 제공해야 할 것이다. (그림 4)는 구현 관점과 타 관점과의 관계를 모형화 한 메타 모형이다.

3.3.4 프로세스 및 배포 관점과 메타 모형

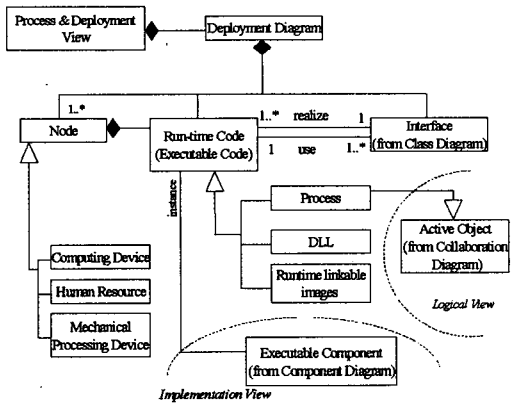
프로세스 관점과 배포 관점은 다중 프로세스에 의해 운영되거나 다중 프로세서 상에서 운영되는 시스템을 개발해야 하는 경우 시스템의 성능, 처리량, 프로세스의 배치 등이 모형화의 주된 관심사이므로 서로 밀접한 연관관계를 맺고 있다고 볼 수 있다. 그러므로 단일 프로세스이거나 단일 프로세서에 의한 시스템을 개발하는 경우 이 두 관점은 모형화 대상에서 제외해도 무방하다.

프로세스 관점은 실제로 시스템이 실행될 때 활성화되는 객체(Object)가 주요 메타 데이터로 활용되며, 배포 관점은 프로세서 또는 데이터베이스를 표현하는 노드가 메타 데이터로 활용된다. 이 두 메타 데이터가 다른 관점과 어떻게 호환되는지 본 연구에서는 프로세스 관점과 배포 관점을 합하여 하나의 메타 모형(그림 5)을 작성하였다.

전개도는 노드와 실행코드(Executable Code)라는 객체와 인터페이스 객체가 주요 구성요소이다. 실행코드는 구성요소도의 세 가지 타입의 컴포넌트 중 실행코드 컴포넌트의 인스턴스로서, 이는 전개도의 모형화 범주가 개발된 시스템의 실제 동작되는 인스턴스 레벨에서의 모형이기 때문이다. 그러므로 프로세스 및 배포



(그림 4) 구현 관점과 메타 모형



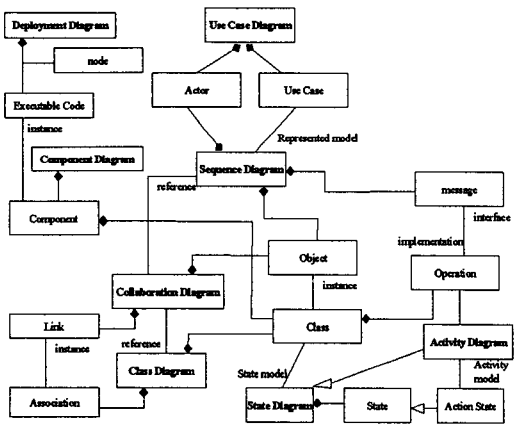
(그림 5) 프로세스 및 배포 관점과 메타 모형

관점은 구성요소도와 밀접한 연관관계를 맺고 있다고 볼 수 있다. 또한 프로세스는 협력도에서는 활성 객체(Active Object)로 달리 불리워질 수 있다. 프로세스는 활성 객체로부터 상속 받은 서브 클래스이다. 그러므로 협력도의 모형정보와도 일관성 여부를 점검하여야 할 것이다.

4. 일관성 검증 규칙 및 CASE 도구 적용사례

4.1 메타 모형 내 일관성 검증 대상 모형요소

3장을 통하여 정의한 메타 모형 중 서로 밀접한 연관관계를 맺고 있는 모형요소들을 중심으로 통합한 메타 모형은 다음 (그림 6)과 같다.



(그림 6) UML 통합 메타 모형

이장에서는 이 통합 메타 모형을 이용하여 중점적으로 파악하여야 할 모형간 일관성 검증 대상 모형요소에 대하여 설명하겠다.

(그림 6)메타 모형에서 주시할 것은 하나 이상의 연결(Connection)을 가지고 있는 모형요소(model element)와 하나 이상의 다이어그램과 연결을 가지고 있는 모형요소이다. 가장 많은 연결을 가지고 있는 모형요소는 클래스(Class)로서 이 모형요소는 클래스도의 구성요소이며, 순서도의 구성요소인 객체(Object)와 연관관계를 가지고 있으며, 구성요소도 컴포넌트의 일부분으로 포함되어야 한다.

행위자(Actor)는 사용사례와 순서도 두 다이어그램의 구성요소이며, 객체(Object)는 순서도와 협력도 두 다이어그램의 구성요소이다. 이 두 모형요소의 경우, 서로 다른 다이어그램에 동일한 의미를 가지고 있는 모형요소가 이용되었을 경우 그 개체가 가지고 있는 정보는 일관성을 유지하여야 한다.

이외에도 3장의 (그림 2), (그림 3), (그림 4), (그림 5) 등 각 관점 별 메타 모형을 검토하면 패키지, 메시지, 연관관계 등 하나 이상의 연결을 가지고 있는 많은 모형요소를 볼 수 있다. 다음 절에서 이들을 분류하였다.

4.2 일관성 검증 규칙

앞 절에서 추출한 규칙들을 유형별로 분류하면 다섯 가지로 분류할 수 있다. 즉, 모든 다이어그램의 모형요소에 관계되는 기본 분석, 객체 모형요소와 관련한 분석, 객체와 객체 간을 연결하고 있는 관계 분석, 객체와 객체 간 정보의 흐름의 일관성 여부를 분석하는 메시지 분석, 기타 다이어그램 간 참조 무결성 분석 등으로 분류하였다.

4.2.1 기본 분석

기본 분석 유형에는 모든 다이어그램에 단독으로 존재가 가능한 객체들의 정체성(Identity)과 각 객체들이 여타 객체와 관계(Relationship)를 맺고 있는가의 두가지가 있다.

여기서 객체란 클래스의 인스턴스로서의 객체만을 의미하는 것이 아니라, 사용사례, 클래스, 패키지, 컴포넌트, 노드 등 그 자체로서만으로도 다이어그램 상에서 존재할 수 있는 다각형의 모형요소를 의미한다. 이들 객체 모형요소는 의미있는 이름을 가짐으로써, 시스템 내에서의 역할을 부여받을 수 있다. 또한 해당 객체가 하나 이상의 다이어그램에서 하나 이상의 객체 모형요소와 관계를 맺고 있다면 그 이름은 모형화 작업을 하는 사용자에게 더욱 중요한 의미를 가지게 될 것이다.

또 한가지는 객체 모형요소가 다른 객체 모형요소와 관계를 맺고 있어야 한다. 관계를 가지지 않고 단독으로 존재하는 객체 모형요소는 향후 시스템이 구축되어 실행 시 어디에서도 사용되지 않는다고 볼 수 있다.

위의 두가지 기본 사항은 전개도를 제외한 모든 다이어그램에서 점검되어야 한다.

4.2.2 객체 모형요소 분석

이 분석은 사용사례도, 순서도, 클래스도, 협력도 내

가지 다이어그램 간 모형정보의 일관성 분석이다. 주로 클래스도의 클래스를 중심으로 타 다이어그램의 모형정보를 검증하고, 결과를 모형을 작성한 사용자에게 경고한다.

● 클래스와 객체

객체와 클래스는 인스턴스와 타입이라는 관계를 가지고 있다. 순서도, 협력도에 나타나 있는 객체가 어떠한 타입(Class)도 정해져 있지 않은 경우, 객체의 완전한 정보를 파악하기가 어렵다.

● 클래스와 컴포넌트

구성요소도의 컴포넌트는 구현 관점에서 연관성이 높은 클래스들이 무리 지어(cluster) 배치되는 가시화되는 개체이다. 즉 모든 클래스들은 반드시 어떤 컴포넌트에 소속해 있어야 한다. 그러므로 모형화 과정에서 클래스가 할당될 컴포넌트가 지정되지 않을 경우, 컴파일이 또는 링크작업이 성공적으로 수행되지 않을 수도 있다.

● 클래스와 패키지

클래스 CA와 패키지 P가 종속관계를 맺고 있을 경우, 이는 클래스 CA가 패키지 P내에 존재하게 될 어떤 클래스와 관계를 맺고 있음을 의미한다. 그러나 클래스 CA와 패키지 P의 관계만 규정지어져 있을 뿐, CA가 패키지 P에 그루핑 되어 있는 어떤 클래스와도 관계를 맺고 있지 않을 경우 CA-P 간 종속관계는 사용자에 의하여 점검되어야 할 것이다.

● 사용사례와 순서도

사용사례는 사용사례도의 구성요소이다. 사용사례는 순서도를 통하여 상세화된 절차를 모형화 할 수 있다. 모든 사용사례 마다 순서도를 반드시 작성할 필요는 없으나, 추출된 각 사용사례를 적절히 표현할 순서도의 모형화는 객체 및 클래스 추출을 위한 중요한 선행작업이라 할 수 있다.

● 행위자(Actor)

행위자는 사용사례도와 순서도에서 모두 사용되는 모형요소로써, 사용사례도에서는 사용사례와 관계를 맺고 있으며, 순서도에서는 객체와 관계를 맺고 있다. 그러므로 행위자와 사용사례와 순서도는 상호 밀접한 관계를 가지고 있다. 즉 행위자 A가 사용사례 Ua, Ub 등을 가지고 있는 사용사례도에서 오직 사용사례 Ua와 유일한 관계를 맺고 있을 경우, 사용사례 Ua의 상세한 절차를 모형화한 순서도 Sua 이외의 순서도에 나타나는 행위자 A는 검토의 대상이 되어야 한다. 즉 Sua 이외의 순서도 Sub에 행위자 A가 표현되었다면, 행위자 A는 순서도 Sub의 사용사례인 Ub와 사용사례도에서 관계를 맺고 있어야 할 것이다.

4.2.3 메시지 분석

이 절에서는 객체 모형요소 사이에서 교환되는 정보인 메시지에 대한 분석에 대하여 언급한다. 객체지향 패러다임에서 클래스 또는 객체와 함께 중요한 개념인 메시지는 모형화 시각에 따라 다른 모습으로 표현된다. 그러므로 다수의 다이어그램에 표현되는 동일 메시지 정보의 일관성 여부는 견고한 모형을 생성하는데 중요한 요소라 할 수 있다.

● 연산과 메시지

순서도 또는 협력도에서 객체 OBa가 링크로 연결되어 있는 객체OBb에게 어떤 행위를 요구하는 메시지 Msg1를 보내는 경우, 이 메시지 Msg1을 받는 객체 OBb는 그에 상응하는 연산 OP-msg1을 가지고 있어야 한다. 객체 OBb가 클래스 Cb의 인스턴스라면 클래스 Cb는 연산 OP-msg1을 소유하고 있어야 하며, Cb의 인스턴스로 정의된 모든 객체는 연산 OP-msg1을 역시 소유하고 있어야 한다.

● 연산과 행동(Action)

특정 클래스가 클래스 외부로부터 받는 자극에 따라 어떤 행동(Action)들을 수행하고 어떤 상태로 변화되는가를 도식화한 다이어그램이 상태도이다. 그러므로 어떤 상태로 변화되기 위해서도 행동이 요구되지만 어떤 상태를 지속적으로 유지하기 위해서도 어떤 행동을 지속적으로 수행하고 있는 것이다. 그런데 이 행동이란 상태의 주체가 되는 클래스가 그가 가지고 있는 연산 들로 이루어져 있으므로 상태도에 정의되는 행동들은 클래스의 연산으로 연계되어야 할 것이다.

● 메시지와 이벤트

UML 메타 모형을 보면 상태도에 정의되는 이벤트는 순서도에 정의되는 메시지의 서브 클래스이다. 즉 메시지 중 일부는 상태도 상에 이벤트로 나타남을 의

미한다. 그러므로 상태도 내에 정의되어 있는 이벤트는 모형 검증 시 점검의 대상이 되어야 한다.

4.2.4 관계 모형요소 분석

협력도에서 클래스의 인스턴스인 객체와 객체를 이어주는 링크(link)는 클래스도에서 클래스와 클래스 사이를 이어주는 연관(Association)의 인스턴스이다. 그러므로 협력도에서 링크가 정의되어 있다면, 클래스도에 연관이 정의되어 있어야 한다. 반대의 경우 클래스도에 정의된 연관이 협력도에 링크로 반드시 정의되어 있을 필요는 없으나, 인스턴스 레벨에서의 모형인 협력도에서 링크가 정의된다면, 이 링크는 반드시 클래스도의 어떤 연관과 관계를 맺고 있어야 한다.

이에 대해서는 협력도 그래픽 모형 편집기 상에서 사용자가 모형화 작업 과정에서 인터랙티브하게 처리하는 방안을 생각할 수 있다.

주어진 협력도 Col1이 있다고 가정한다. 협력도 Col1에는 객체 Ob1과 Ob2가 존재한다. 이 객체 Ob1과 Ob2가 상호 관련이 있어 링크 Lnk1을 연결하고자 할 때, 모형 편집기는 세 가지로 대응할 수 있다.

첫째 객체 Ob1이 클래스 C1의 인스턴스인 경우, 클래스 C1이 맺고 있는 모든 연관정보를 사용자에게 보여주고 그 중 하나를 선택하도록 한다.

둘째 객체 Ob1, Ob2가 각각 클래스 C1, C2의 인스턴스인 경우, 클래스 C1, C2가 상호 연관관계를 맺고 있다면 그 연관정보를 링크 Lnk1에 상속한다.

셋째, Lnk1으로 이어질 객체 Ob1, Ob2 모두 클래스가 지정되어 있지 않은 경우, 클래스도에 정의되어 있는 모든 연관을 보여주고, 그중 하나를 선택하도록 한다. 사용자에 의해 연관이 선택되면, 객체 Ob1, Ob2는 선택된 연관의 양 끝에 정의되어 있는 클래스들의 인스턴스로 정의된다. 사용자가 어떤 연관도 선택하지 않은 경우, 새로운 링크가 생성된다.

첫번째와 두번째 경우는 모형 편집기에 의해 구현이 가능하나, 세번째 사용자가 어떠한 연관도 선택하지 않은 새로운 링크가 생성되는 경우는 모형 일관성 검증의 대상이 된다.

4.2.5 객체의 참조 무결성 분석

● 구성요소도 내의 인터페이스 클래스

구성요소도(Component Diagram)의 컴포넌트는 컴파일, 링크 및 실행을 위한 시스템의 포장(Packaging) 단위로 볼 수 있다. 컴파일, 링크 시 다른 모듈에 소속해 있는 기능을 호출할 경우, 이를 참조할 수 있는 인터페이스가 존재해야 작업을 성공적으로 수행할 수 있다. 이를 지원하는 모형 메커니즘이 인터페이스 클래스이다. 그러므로 실제로 이 패키징 작업을 시작하기 전에 구성요소도를 통하여 컴포넌트와 타 컴포넌트 간 인터페이스를 모형화 한다.

이때 인터페이스 클래스는 클래스도의 어딘 가에 정의되어 있어야 한다. 구성요소도에 정의되어 있는 인터페이스 클래스가 클래스도에 정의되어 있지 않다면, 이 클래스 모형을 구현단계까지 확장해 나갈 경우 실패할 수 있다.

또한 컴포넌트와 인터페이스 클래스가 실현(Realization-dependency)관계에 있다면, 그 인터페이스 클래스는 관계를 맺고 있는 컴포넌트 내에 할당되어 있는 클래스들 중 한 클래스와 반드시 연관관계를 맺고 있어야 한다.

● 상태도 내의 객체

상태도는 클래스의 상태와 상태 변화를 도식화하는 다이어그램이다. 이 다이어그램에 때로 상태를 유지하기 위하여, 또는 상태가 변화하는 과정에서 수행되는 행동 중에 타 객체에 메시지를 전달하는 과정이 포함되기도 한다. 이때 상태도에 객체가 표현될 수 있는데, 이 경우 객체를 대표하는 클래스와 상태도의 주체인 클래스는 상호 어떤 관계를 맺고 있어야 한다. 그러므로 상태도에 객체가 표현되어 있을 경우 모형 검증의 대상이 된다.

● 전개도 내에서의 활성 객체

전개도에는 개발 후 설치될 시스템의 단말 또는 서버를 의미하는 노드(Node)가 주요 모형요소로 사용되는데, 필요에 따라 동적인 프로세스로 표현되는 활성 객체가 표현되기도 한다. 이때 활성 객체는 여타 클래스도의 클래스들 중 어느 한 클래스의 인스턴스이어야 한다.

5. CASE 도구 적용사례

3장 및 4장에서 도출된 모형간 일관성 규칙은 DEBUT라고 하는 객체지향 CASE 도구의 구현 시 적용되었다. DEBUT는 Design by UML Tool의 약자로서 객체지향

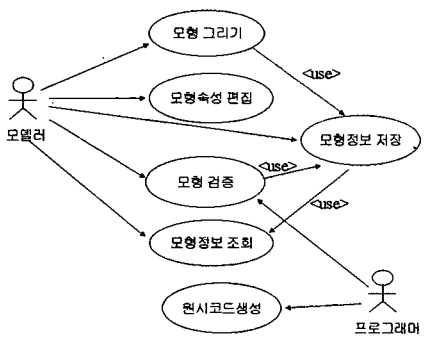
소프트웨어 시스템 모델링 도구이다. 시스템 분석가 및 소프트웨어 개발자들이 소프트웨어 개발 전반에 걸쳐 사용자 요구사항 추출 및 시스템 분석 설계 시에 활용하는 도구로서, 객체지향 분석 설계 표기법인 UML 1.1의 표기법, 구문, 의미를 준수하고 있다. 5장에서는 사용자가 본 CASE 도구를 활용하는 사용사례를 살펴보고, 사용사례 중 모형 검증기능 구현을 위한 순서도를 모형화 한 후, 그에 따른 실제 구현 화면에 대하여 설명한다.

5.1 시스템 사용사례

CASE 도구를 활용하여 시스템을 개발하는 경우 CASE 도구에 대한 사용사례를 사용사례도를 사용하여 모형화하면 (그림 7)과 같다.

시스템을 사용하는 사용자(액터)는 분석, 설계를 수행하는 모델러와 특정 프로그래밍 환경과 연결하여 CASE 도구를 사용하는 프로그래머로 나누어 볼 수 있다. 모델러는 모형 그리기, 각 모형요소의 속성 편집하기, 모형 정보 조회, 모형 검증 등을 위하여 CASE 도구를 사용하고, 프로그래머는 모형 검증 또는 원시코드 생성 등을 위하여 CASE 도구를 사용할 수 있다. 또한 모형 그리기를 통하여 생성된 모형정보를 저장하여 두었다가 화면에 다시 재생하거나, 다른 모형에 재사용할 수도 있다.

먼저 '모형 그리기' 기능을 만족하기 위해서는 다이어그램 편집기 또는 모형 편집기가 요구된다. 이 다이어그램 편집기는 모형요소 아이콘과 그래픽 편집 기능을 가지고 있어 사용자의 작업을 통하여 모형요소들을 생성한다. 모형 편집기를 통하여 생성된 위치정보와 표현 정보(representation information)등으로 구성된



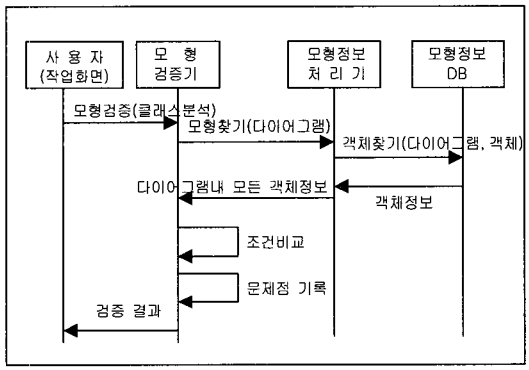
(그림 7) CASE 도구 사용사례

그래픽 정보는 객체 고유 정보와 함께 텍스트 정보로 변환되어 모형요소 명세로써 메모리에 저장된다.

위의 사용사례들 중 사용자가 모형의 일관성을 검증하기 위하여 CASE 도구의 '모형 검증' 기능을 이용할 경우를 순서도(Sequence Diagram)로 모형화하면 (그림 8)과 같다.

이와는 별개로 사용자는 자신이 작성한 모형의 일관성 여부를 수시로 점검할 수 있다. 이때 기존에 구축되어 있는 모형 검증 규칙 정보에 근거하여 모형요소 명세의 일관성 여부를 검증하게 된다.

모형 요소의 검증 결과는 사용자가 작업하는 컴퓨터의 스크린에 별개의 창을 통하여 볼 수 있다.



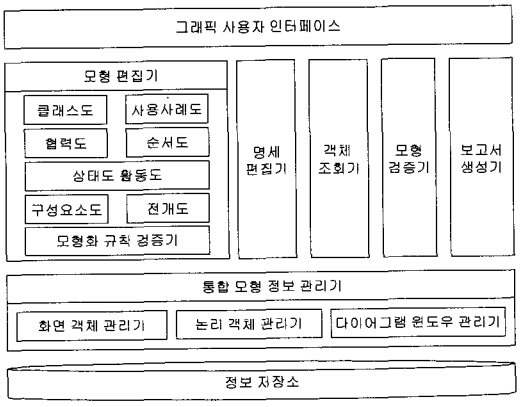
(그림 8) 모형 검증 순서도

5.2 시스템 구성

위의 사용 사례들을 만족하는 시스템을 구성하기 위하여서는 몇가지 기본 서브 시스템이 요구된다. 먼저, 가장 중요한 그래픽 모형 편집기와, 모형을 구성하는 각 구성요소의 속성값을 지정하고 변경할 수 있도록 모형요소 편집기가 요구된다. 둘째, 모형정보를 체계적으로 보관하고 있어서 다양한 형태의 검색을 용이하게 할 모형정보 관리기가 요구된다. 셋째, 모형 검증 규칙을 포함하여, 모형정보의 일관성 여부를 검증할 모형 검증기가 요구된다.

그 외에도 CASE 도구가 갖추어야 할 기능을 제공하기 위하여, 사용자의 모형화 결과를 다양한 형태로 출력할 수 있도록 하는 보고서 생성기, 개발환경에 적합한 프로그래밍 언어로 변환해 주는 원시코드 생성기, 모형내에 존재하는 모형요소 및 객체정보를 조회할 수 있는 모형조회기 및 다이어그램 조회기 등의 서브 시스템을 갖출 수 있다.

DEBUT의 구성은 7개 다이어그램을 통한 모델링 작업을 지원하는 그래픽 모형 편집기, 사용자의 작업 내용을 시각적이며 계층적으로 보여주고 관리해주는 프로젝트 관리기, 사용자의 입력에 의한 그래픽 정보와 모형화 대상 시스템에 유일하게 존재하게 되는 논리 정보를 통합 관리하며 다른 서브 시스템에 정보 서비스를 담당하는 통합 모형 정보 처리기, 통합 모형 정보 처리기를 통하여 사용자의 모형 정보를 조회하거나, 분석해주는 모형조회/검증기, 보고서 생성기 등으로 이루어져 있다. (그림 9)는 DEBUT의 시스템 구성도이다.



(그림 9) DEBUT 시스템 구성

5.3 시스템 주요 기능

DEBUT의 모형 편집기는 클래스도, 사용사례도, 협력도, 순서도, 상태도, 구성요소도, 전개도 등 UML의 7개 다이어그램을 시각적으로 편집할 수 있도록 지원하는 시각 편집기이다. 이 편집기의 주요 기능은

- UML 모형요소 생성
- 모형요소 명세 편집
- 다이어그램 확대 및 축소
- 격자, 모형요소 정렬 및 색상 변경
- 모형요소의 자르기, 붙이기, 복사, 삭제 기능
- 대화식 모형 규칙 검증 기능

등이며 특히 대화식 모형 규칙 검증기는 각 다이어그램의 객체 생성 규칙 및 객체 간 연결규칙을 UML의 표기법 및 규칙에 근거하여 사용자의 모델링 작업을 인터랙티브하게 제어하도록 하여, 사용자가 UML의 모델링 규칙에 어긋나는 모형을 작성하지 않도록 하였다.

5.4 모형 검증기 설계 및 구현

모형 검증기는 CASE 도구의 도구 메뉴의 일부분으로 구현되었다. 검증 유형에 따라 기본 분석, 객체 모형요소 분석, 메시지 분석, 관계 모형요소 분석, 객체 참조 무결성 분석으로 세분화 하여 사용자는 메뉴에서 원하는 검증 유형을 선택할 수 있다. 본 절에서는 각 메뉴별로 모형 검증의 대상이 되는 모형요소를 나열하고, 해당 모형요소가 가지고 있는 특성 값 및 조건을 이용하여 검출할 수 있는 오류의 유형 및 사용자에게 나타내는 메시지 유형을 나열하였다.

- 기본 분석

대 상 모형요소	오류 유형	관련 메시지
다각형의 모형요소(클래스, 객체, 패키지 등)	모형요소가 이름을 가지고 있지 않는 경우	객체가 의미있는 이름을 가지고 있지 않다.
	모형요소가 단 하나의 관계도 가지고 있지 않은 경우	객체가 아무런 관계도 가지고 있지 않다.

- 객체 모형요소 분석

대 상 모형요소	오류 유형	관련 메시지
모든 객체	객체가 어느 클래스의 인스턴스인지 알 수 없는 경우	객체의 타입이 지정되어 있지 않다.
모든 클래스	클래스가 컴포넌트에 할당되어 있지 않은 경우	클래스가 어떤 컴포넌트에도 소속되어 있지 않다.
패키지와 종속관계를 가지고 있는 클래스	패키지내에 소속되어 있는 클래스들 중 어떤 클래스와도 관계를 가지고 있지 않은 경우	패키지와의 종속관계를 입증할 수 없다.
사용사례	사용사례를 모형화한 순서도가 존재하지 않는 경우	사용사례에 대한 순서도가 정의되어 있지 않다.
사용사례와 종속관계를 가지고 있는 행위자	사용사례를 모형화한 순서도에 행위자가 정의되어 있지 않은 경우	행위자가 적절한 순서도에 정의되어 있지 않다.

- 메시지 분석

대 상 모형요소	오류 유형	관련 메시지
메시지	메시지를 받는 객체의 타입 클래스가 동일한 이름의 연산을 가지고 있지 않은 경우	메시지에 해당하는 연산이 정의되어 있지 않다.
행 동 (Aciton)	행동의 주체가 되는 객체의 타입 클래스가 동일한 이름의 연산을 가지고 있지 않은 경우	행동(Action)에 해당하는 연산이 정의되어 있지 않다.

● 관계 모형요소 분석

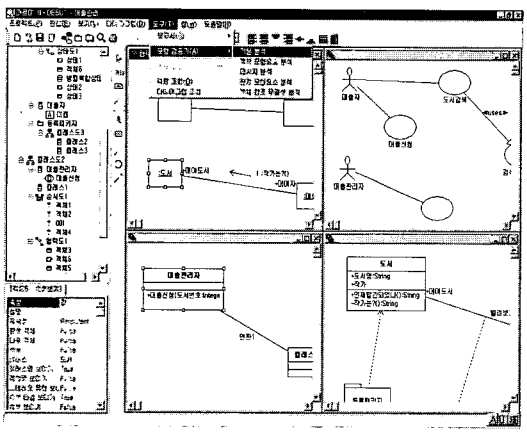
대 상 모형요소	오류 유형	관련 메시지
링 크	링크가 연관(Association) 정보를 가지고 있지 않은 경우	링크 정의가 불완전하다.

● 객체 참조 무결성 분석

대 상 모형요소	점검사항	오류 유형
컴포넌트와 종속관계를 가지고 있는 인터페이스 클래스	어느 클래스 도에도 소속되어 있지 않거나 여타 다른 클래스와 관계를 가지고 있지 않은 경우	인터페이스 클래스 정의가 불완전하다.
상태도에 존재하는 객체	이 객체의 타입클래스가 상태도의 주체인 클래스와 아무런 관계도 가지고 있지 않은 경우	객체정의가 불완전하다.

이밖에 클래스의 연산 중 visibility = private으로 되어 있는 연산을 메시지로 활용하려는 경우에 대한 오류 점검이나 클래스가 결정된 객체들 간의 링크의 경우 자동 association 설정 등은 모형화 작업 시에 지원될 수 있도록 구현하였다.

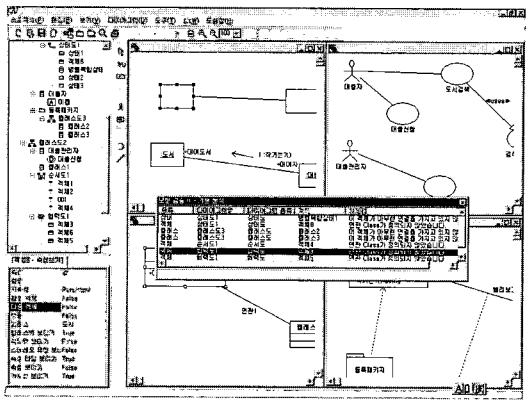
(그림 10)은 사용자에 의해 작성된 다이어그램 화면과, 모형 검증기를 통하여 분석할 수 있는 모형 분석 유형에 대한 서브 메뉴를 보여주고 있는 실행 화면이다.



(그림 10) 모형검증기

(그림 11)은 모형 검증기 실행 화면이다. 주도구 박스에서 도구 메뉴 선택 후 모형 검증 유형 중 기본 분석을 선택하여 모형을 분석한 화면으로, 클래스도에 정의된 클래스가 어떤 클래스와도 관계를 맺고 있지 않음을 지적하고 있다.

모형 검증기의 분석 결과 중 한 라인을 선택한 후 더블 클릭하면 지적된 모형요소를 포함하고 있는 모형 편집기 화면으로 이동하며, 문제가 되는 객체의 경계선에 빨간색 점을 넣어 식별이 쉽도록 하였다.



(그림 11) 모형 검증기 기본분석 결과

6. 결 론

90년대 이후 보다 견고한 객체지향 패러다임을 구축하려는 노력으로 객체지향 시스템을 표현하는 하나의 통일된 모형 언어인 UML이 업계 표준화라는 결실을 맺었다. 이에 따라 이 표준 모형 언어 UML이나 UML을 지원하는 CASE 도구를 활용하여 신규 소프트웨어 시스템을 개발하거나, 기존 시스템(Legacy System)을 객체지향 시스템으로 전환하려는 노력이 꾸준히 전개되고 있다. 한편으로는 이 언어의 보다 효율적인 활용을 도모하기 위하여 객체지향 시스템의 개발과정과 활동, 그에 따른 결과물(Documentation) 등을 정의하여 이를 보급하려는 움직임도 따르고 있다.

다양한 각도에서의 소프트웨어 시스템 표현을 지원하는 많은 모형(Model)과 그에 따른 다이어그램(Diagram)들이 제공되고 있으나, 그 다양한 모형들이 종국에는 하나의 시스템으로 통합되어야 한다는 점에서 모형 정보간 일관성 확보는 개발과정(Software Development Lifecycle)에서 매우 중요하게 다루어져야 할 작업이다. 분석 및 설계 단계에서 확정된 모형들이 향후 실제 개발될 시스템의 추상화된 표현이라는 점에서 볼 때, 견고한 시스템은 견고한 모형으로부터 도출되기 때문이다.

본 연구에서는 객체지향 모형화 언어 UML의 다양한 모형들이 상호 일관성을 유지할 수 있도록 하기 위하여 UML 모형 간 일관성 검증 기법을 연구하고 이를 객체지향 CASE 도구 DEBUT에 적용하였다. 먼저 소프트웨어 아키텍처를 4+1 관점으로 분류하고 각 관점별 활용 가능한 UML의 다이어그램들을 배분한 후, UML notation 가이드에 정의되어 있는 UML 모형요소 메타 모형을 기반으로 하여 각 관점별 메타 모형을 정의하였다. 그리고 이 메타 모형으로부터 점검 대상 모형요소와 일관성 유형을 추출하였다. 모형 검증기는 UML 지원 객체지향 CASE 도구 DEBUT에서 사용자의 모형화 결과를 검증하기 위한 부기능으로 구현되었다.

향후 메타 모형은 지속적으로 정제되어야 할 것이며, 아직까지 개념만 정의되어 있고 용도 및 활용 방법 등이 구체화되지 않고 있는 협력도의 구성요소인 패턴(Pattern)도 고려대상이 되어야 할 것이다. 또한 현재까지는 97년에 제정된 UML 1.1이 유지되고 있으나, 각계의 피드백을 수렴하여 개정과 함께 OMG에 상정되어 있는 UML 1.3[13]에 따르면, UML 모형요소 객체의 속성을 정형화된 형태로 정의할 수 있도록 지원하는 언어인 OCL(Object Constraint Language)에 대한 많은 수정이 가해졌다. 이 OCL이 모형요소 정의 언어로 정착되면 OCL을 통한 모형요소의 제약사항 및 모형 간 일관성 정의도 연구되어야 할 것이다.

참 고 문 헌

[1] 전진옥 외, 정보통신용 시스템 개발방법론 구축에 관한 연구, 연구보고서, 한국전자통신연구원, 1998. 11

[2] Booch, Grady, Object-Oriented Design with Application, Benjamin-Cummings, 1991.

[3] Jacobson, Ivar, Object-Oriented Software Engineering : A Use Case Driven Approach, Addison-Wesley, 1992.

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

[5] Rational, Unified Modeling Language(Version 1.1), Rational Software Corp., 1997.

[6] Rational, Rational Unified Process, Rational Software Corp., 1998.

[7] Clements, Paul C. and Northrop, Linda M., Software Architecture : An Executive Overview, Technical Report, Software Engineering Institute, Carnegie Mellon University, 1996.

[8] Kruchten, Philippe, The 4+1 View Model of Architecture, IEEE Software, November, 1995.

[9] 김치수, 진영진, 객체지향 분석의 완전성과 일관성 검증을 위한 툴의 설계, 한국정보처리학회 논문지 제4권 제10호, 1997, pp.2453-2460

[10] 박진호, 김수동, 류성열, UML 다이어그램들 간의 일관성 검증 방법, 한국정보과학회 봄 학술발표논문집, Vol.25 No.1, 1998, pp.524-526

[11] Shaw, Mary, et al., Software Architecture, Prentice Hall, 1996.

[12] Booch, Grady, Software Architecture and The UML, Rational Software Corp., 1999.

[13] Rational, UML Metamodel Abstract Syntax v1.3 R20 Changes, Rational Software Corp., 1999.

이 선 미

e-mail : lsm@etri.re.kr

1985년 한국외국어대학교 사회과학대학 신문방송학과(학사)

1999년 충남대학교 대학원 컴퓨터과학과(석사)

1986년 7월 현재 한국전자통신연구원 실시간컴퓨팅연구부 선임연구원

관심분야 : 소프트웨어 공학, 객체지향 모델링, 개발방법론

전 진 옥

e-mail : jojeon@etri.re.kr

1984년 한국외국어대학교 교육학부 독일어(학사)

1987년 미국 조지아 주립대학원 정보시스템(석사)

1995년 한국외국어대학교 경영정보학과(박사)

1986년~1987년 미국 INTEC 연구소(위촉연구원)

1987년~현재 한국전자통신연구원 실시간컴퓨팅연구부장(책임연구원)

관심분야 : 소프트웨어 공학, 실시간 시스템

류 재 철

e-mail : jcryou@esperosun.chungnam.ac.kr

1985년 한양대학교 산업공학과 (학사)

1988년 Iowa State University 전산학과(석사)

1990년 Northwestern University 전산학과(박사)

1991년~현재 충남대학교 컴퓨터과학과 부교수

관심분야 : 컴퓨터통신보안, 전자상거래, 분산처리