객체 모델의 프로토타이핑을 위한 명세 및 시뮬레이션 환경

정 란[†] · 김 정 아[††] · 문 충 렬[†††] · 김 정 두[††††]

요 약

객체 모델링을 통해 초기에 사용자의 요구를 보다 명확하게 정의할 수 있고 그 결과를 바탕으로 보다 완전한 프로그램을 얻을 수 있으나, 모델 자체만으로 사용자 요구 사항과의 부합 여부를 쉽게 판단할 수 없다. 본 연구에서는 사용자와의 교류 수단으로의 객체 모델링과 검증이 가능한 형식 명세의 통합을 통해 이해하기 쉽고 구축하기도 쉬우며 검증할 수 있는 모델을 얻을 수 있게 하였다. 구축한 객체 모델을 구현 이전에 시각적 실행 환경에서 미리 검증해보므로써 개발 초기에 보다 정확한 모델을 얻을 수 있게 한다. 이를 위해 본 연구에서는 OOSA(Object-Oriented System Analysis) 방법론으로 구축한 객체 모델을 형식 명세 언어인 VDM으로 변환하는 규칙을 제안하였다. 이미 검증 과정을 거친 형식 명세 언어인 VDM으로 변환한 후 이를 바탕으로 구축한 실행 지원 환경을 통해 객체 모델의 프로토타이핑 과정을 수행할 수 있게 지원한다.

Specification and Simulation Environment for Prototying the Object Model

Lan Jung[†] · Jung A Kim[††] · Chung Ryeal Moon[†††] · Jung Doo Kim[††††]

ABSTRACT

Object modeling has been considered as an efficient technique for eliciting user requirements and communicating between developers and customers. But model itself is not easy to understand what result will be after coding and whether it will be meet with the requirements of customers. In this paper, we developed the environment for visualization of object model for validating with requirement at the early stage. Therefore, we defined correct and complete rules which can transform the object model, the deliverables of Shlaer/Mellor's method, into a formal specification language of VDM(Vienna Development Methods) with a mathematical basis. This basis provides the means of proving that a specification is realizable and proving properties of a system. Therefore, the completeness, preciseness of object model can be verified by proving the transformed VDM specification and prototyping by constructing a visualization supporting environment.

1. 서 론

최근 소프트웨어 시스템의 적용 범위가 급격히 확장됨에 따라 매우 복잡해지는 추세이다. 이는 소프트웨어 개발 비용의 증대뿐만 아니라 유지 보수에 필요한 노력의 증가를 초래한다. 따라서 개발 노력과 유지 보수 비용을 줄이기 위하여 개발 초기 단계에서 많은 투자가 필요하다. 왜냐하면 분석과 설계 단계에서 발견되는 오류를 수정하는 것이 구현 및 유지 보수 단

† 종신회원:삼척 산업대학교 전자계산학과 교수
†† 정 회 원:관동 대학교 컴퓨터 교육과 교수
††† 비 회 원:현대 전자 소프트웨어 연구소 연구원
†††† 정 회 원:대구효성카톨릭대학교 전자계산학과 교수
논문접수:1997년 1월 9일, 심사완료:1997년 5월 6일

계에서의 수정보다 적은 노력으로 가능하기 때문이다[1, 3, 4]. 많은 연구[8, 9, 11, 12, 14]에서 객체 모델링 방법론을 개발 초기 단계의 부정확한 산출과 불충분한 모델 구축의 문제를 해결할 수 있는 방법론으로 간주하고 있다. 사용자의 요구를 정확하게 표현하고, 분석 단계에서 발생할 수 있는 오류를 미리 방지하여 고객의 요구를 정확하게 반영할 수 있는 방안이기 때문이다. 분석, 설계, 그리고 구현의 단계에서 표준화되고 일관된 단일 객체 모델을 기반으로 하고 있어, 각 단계별 전이가 자연스럽고 추적이 용이하다. 그러므로 초기의 정확한 요구 모델의 구축은 올바른 소프트웨어를 얻을 수 있는 기반이 된다.

객체 모델링 방법론은 실세계를 자연스럽게 모델링 하므로써 고객의 요구 사항을 쉽게 추출하게 하며, 분석가와 고객 양측에게 의사 교류를 용이하게 한다[1, 3, 4]. 객체 지향 설계 단계에서 사용될 수 있는 기반으로 객체 모델은 개발 단계간의 자연스러운 전이를 가능하게 한다[5]. 그러나 고객이 이해 하기 쉬운 객체 모델이라도 새로운 모델링 도구를 익혀야 하고 모델 자체만으로는 그 정확성을 확인하기 힘들다. 이를 보완할 수 있는 방안으로 실행 가능한 요구 명세 또는 실행 가능한 모델을 개발 과정에 적용하는 것이다. 단지 문서화나 다음 단계 작업의 입력물로써만의 모델이 아닌 프로그램이 실행을 통해 그 결과를 알수 있듯이 모델을 실행하여 모델의 정확성을 검증하는 것이다. 본 연구에서는 기존의 모델을 실행 가능한 명세 언어나 실행 가능한 모델로 변형하기 보다는 모델을 기반으로 객체들의 상태 변화를 중심으로 한 시뮬레이션 환경을 구축하고자 하였다. 문서화와 개발의 정형화를 위해 우선 객체 모델링을 통해 문제를 분석하고 객체 모델을 구축한다. 이를 시뮬레이션 하기 위해 먼저 내부적인 기술(description)로 변환한다. 시뮬레이션의 입력으로 사용한 객체 모델에 대한 내부 표현으로 새로운 언어 구조를 정의하기 보다는 본 연구에서는 기존의 상태 기반 형식 명세 언어를 적용하였다. 이를 위해 객체 모델을 기존의 형식 명세 언어인 VDM으로 변형하는 규칙을 정의하였다. 규칙에 따라 VDM으로 변형 후 내부 표현을 바탕으로 모델의 수행에 대한 시각적 환경을 이용해 사용자에게 확인시키기 위한 프로토타이핑 방법으로 모델의 시뮬레이션 환경을 구현하였다.

본 논문의 구성은 2장에서는 객체 모델링 방법으로 선택한 Shlaer/Mellor의 OOSA(Object-Oriented System Analysis), 그리고 프로토타이핑에 대하여 기술한다. 3장에서는 객체 모델을 VDM으로 변환하는 규칙들을 제안하고 4장에서는 검증된 모델을 사용자에게 확인하는 수단으로 시뮬레이션 방법에 대하여 설명한다. 5장에서는 결론 및 향후 연구 과제를 기술한다.

2. 기반 연구

2.1 Shlaer/Mellor의 OOSA 방법론

Shlaer/Mellor 방법론은 산업계에서 소프트웨어 개발에 적용할 수 있는 잘 정의된 체계적 방법으로 평가받고 있다. 방법론의 가장 중요한 특징은 대형 프로젝트를 도메인으로 분할한 후 각 도메인별로 분석 과정을 진행하므로써 복잡도 관리의 효과적 방법을 제공한다는 것이다. Shlaer/Mellor의 OOSA를 통해 어플리케이션 도메인을 분석하는 것이 다음 단계이다. 이 단계에서는 도메인의 객체와 객체들 사이의 관련성을 표현하는 정보 모델(Information Model), 각 객체의 행위와 객체들 간의 상호 작용 관계를 보여주는 상태 모델(State Model), 상태 모델이 요구하는 처리 과정을 보여주는 행위 자료 흐름 다이어그램(Action Data Flow Diagram:ADFD)을 만들게 된다[5, 14].

정보 모델은 문제의 기본 객체와 객체들간의 관련성을 추출하여 표현하므로써 어플리케이션의 구조적 모습을 보여준다. 그러므로 정보 모델의 궁극적 목적은 문제 영역에 속한 객체를 추출하여 표현하는 것이다. 객체와 관련성의 식별을 완료하면 동적 모델링 단계에서 각 객체에 대한 상태와 상태 전이의 개념으로 객체의 동적 특성을 정형화하는데, 상태 모델은 정보 모델을 구성하는 각 객체에 대해 모델링 한다. 상태는 객체가 가질 수 있는 물리적, 논리적 특징과 법칙을 추상화한 것으로 다른 상태의 행위적 관점에서의 객체를 표현한다. 객체가 제공하는 오퍼레이션을 이벤트로 간주하고, 이벤트는 객체의 현재 상태에서 또 다른 상태로의 전이를 유발시키는 원인이 된다. 객체는 임의의 상태를 이루거나, 유지하기 위해 필요한 처리(processing), 질의(query), 다른 객체와의 상호작용(interaction) 등의 행동들(actions)을 수행하게 된다. 상태, 전이 규칙, 이벤트, 행동들의 집합으로 구성

되는 상태 모델은 객체의 라이프 사이클(Lifecycle)을 정형화한 모델로 상태 전이도(State Transition Diagram:STD)와 상태 전이표(State Transition Table: STT)로 모델링 된다.

2.2 프로토타이핑 기법

소프트웨어 개발 과정에서 검증의 역할과 함께 사용자로부터의 요구 사항의 확인 작업 역시 중요한 과정이다. 이를 위해서 JAD(Joint Application Development), RAD(Rapid Application Development)등의 연구에서 프로토타이핑의 중요성이 커지고 있다. 프로토타입은 향후 개발·완성하고자 하는 시스템의 특성을 갖는 초기의 부분적인 모델이다. 이러한 프로토타입의 특성은 최소한의 노력으로 시스템을 만들어서 사용자의 요구를 확인할 수 있으며, 개발될 시스템의 기초가 될 수 있다[2].

소프트웨어 시스템에 대한 프로토타이핑 기법은 프로토타이핑 언어를 사용하여 개발하는 방법, 또는 형식 기법을 사용하여 개발하는 방법이 있다[2, 7]. 프로토타이핑 언어를 사용하는 방법은 시스템에 대한 분석 모델을 프로토타이핑 언어로의 변환이 필요하다. 따라서 분석 모델과 프로토타이핑 언어가 서로 일관되지 못한다면 분석 모델이 프로토타이핑 언어로 변환될 때 정보 손실의 위험을 내포하고 있다. 형식 기법에서 제공하는 프로토타이핑은 형식 기법 자체에서 프로토타이핑을 제공한다. Prolog나 LISP 같은 논리 언어를 사용하여 명세와 동시에 실행 가능하게 한다[11]. 그러나 형식 기법을 사용한 프로토타이핑은 형식 기법의 기반인 수학 이론 그 자체가 이해하기 어렵기 때문에 많은 노력과 비용을 필요로 한다. 따라서 형식 명세에 익숙하지 못한 사용자에게는 이해하기 어렵다는 단점이 있다.

위의 문제점을 해결하기 위해서 Embley는 객체 지향 시스템 개발에 대한 사용자와의 확인을 위하여 CASE 도구인 IPOST(Interactive Prototyping Object-Oriented Specification Tool)와 CASE 도구 자체에 포함된 분석 및 명세 방법을 개발하였다. IPOST는 OSA (Object-oriented Analysis Model)을 확장한 OSS(formal Object-Oriented Specification model)을 이용하여 시스템을 명세하고, 이 명세를 이용하여 프로토타입을 자동으로 생성하여 사용자와 대화식으로 진행하여 사용자의 요구 사항을 확인한다[12]. 그러나 IPOST에서 사용하는 모델과 프로토타이핑 기법은 일관성이 있지만 사용자는 새로운 분석 모델인 OSS를 배워야 한다는 단점이 있다.

이에 본 논문에서는 사용자의 요구 사항 확인을 위하여 이미 OOSA 방법론에 의해 만들어진 분석 모델을 시뮬레이션 할 수 있는 도구를 개발하고자 한다.

3. 변환 규칙

시뮬레이션에 필요한 객체 모델의 구문적 의미적 정보를 표현하기 위하여 객체 모델 명세 도구를 정의하였다.

3.1 개 요

시뮬레이션을 위해 사용자가 모델링 한 분석 모델을 텍스트 형태로 기술하기 위한 내부 표현 방식을 정의하였다. 이 표현 방식을 제공하는데 있어 단순한 텍스트 기호를 사용하기 보다는 기존의 형식 명세 언어인 VDM[6]으로 작성하였다. 즉, 본 연구에서는 분석 모델에 대한 BNF 형식의 새로운 언어를 정의한 것이 아니라, 기존의 형식 명세 언어인 VDM을 적용하였다.

3.2 정보 모델 및 동적 모델에 대한 VDM 명세 변환 규칙

OOSA의 정보 모델을 VDM의 상태 도메인으로 변환하기 위해 정보 모델에 표현된 각 객체와 관련성을 VDM의 상태 요소로 변환하기 위한 8개의 규칙을 정의하였다. 또한 OOSA의 동적 모델을 VDM의 오퍼레이션 도메인으로 변환하기 위하여 7개의 규칙을 정의하였다.

변환 규칙 1:정보 모델의 객체와 속성은 각각 VDM의 Record와 이를 구성하는 필드로 변환한다. 속성들 중 참조 속성은 속성 변환 규칙이 아닌 관련성 변환 규칙을 따른다. 이는 참조 속성이 객체들간의 관련성 정보를 포함하기 때문이다.

변환 규칙 2:한 객체의 식별자가 하나 이상으로 구성되는 경우에는 복수 개의 식별자들을 필드로 갖는 특별한 식별자 Record를 생성한다. 새로 생성한 Re-

cord 이름에 "ID"를 붙여서 이름을 부여하고 원래 객체 필드의 식별자들을 새로 생성한 Record 이름으로 대체한다. 이는 하나의 ADT에는 가능하면 하나의 도메인을 갖도록 하는 것이 추상화의 관점에서 바람직하다는 자료 추상화 이론에 기반한 것이다.

변환 규칙 3:정보 모델의 관련성은 기본적으로 VDM의 Map타입으로 변환하고 관련성에 대한 의미를 표현하는 Invariant로 생성한다.

객체간의 관련성은 기본적으로 양방향의 특성을 갖는다. 즉 두 객체 A, B 사이의 관련성이 존재한다면 한 객체 A로부터 관련된 연관 객체 B를 찾을 수 있으며, 이 관련성에 의해 반대로 연관 객체 B로부터 관련된 객체 A를 식별할 수 있다. 이런 관련성의 특성은 VDM의 상태 도메인 Map으로 변형해야 한다.

변환 규칙 4:1:1의 경우 두개의 Map 타입으로 변환하며 관련성의 주체인 객체가 정의역(Domain)이 되고 연관 객체는 치역(Range)이 되며, 정의역에 관련성이 갖는 특성의 조건을 적용해서 얻어진 치역은 오직 하나여야 한다.

변환 규칙 5:1:M의 경우 두개의 Map 타입으로 변환하며 변환될 Map 타입 중에 정의역이 다(M)이고 치역이 1인 경우에는 1:1의 변환 규칙과 동일하다. 그러나 정의역이 1이고 치역이 M인 Map 타입은 치역의 객체는 집합으로 정의되고, 정의역에 제약을 가할 때 얻는 치역은 오직 하나이도록 변환해야 한다. 그 이유는 다쪽의 Map 타입의 경우는 치역이 여러 값일 수 있으므로 Map 타입의 일대일 대응 제약에 위배되어 관련성을 나타낼 수 없다. 그러므로 새로운 Map 타입의 치역을 Set 타입으로 정의하여야 한다.

변환 규칙 6:M:N의 경우에는 먼저 관련성 정보를 치역이 집합으로 정의되는 두개의 Map 타입으로 변환한다. 관련 객체 자체를 Record로 변환할 때, 이 두개의 Map 타입을 Record의 필드로 구성한다. 연결속성이 정의되어 있다면 이들을 변환된 Record의 필드로 변환한다.

변환 규칙 7:1:M의 경우 M에 해당하는 객체의 참조 속성은 M에 해당하는 객체를 정의역으로, 1에 해당하는 객체를 치역으로 가지는 Map 타입으로 변환한다. 그리고 이 Map 타입은 다에 해당하는 객체를 변환한 Record의 한 필드로 변환된다.

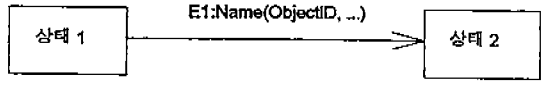
변환 규칙 8:두 객체가 상위/하위 타입으로 관련성을 갖는 경우, 하위 타입은 Record 타입으로 변환되고 하위 타입의 Record에는 상위 타입의 속성이 필드로 포함된다.

다음은 동적 모델의 변환 규칙이다. VDM의 오퍼레이션 영역은 오퍼레이션 인터페이스인 시그네이처와 오퍼레이션 처리에 필요한 선행조건과 처리 후의 결과에 대한 후행 조건으로 구성된다. 동적 모델의 상태는 오퍼레이션의 선행/후행 조건 영역에 대응되며 이벤트와 이벤트 데이터는 오퍼레이션 인터페이스에 대응된다. 이러한 부분의 변환은 다음의 변환 규칙을 따른다.

변환 규칙 9:동적 모델의 이벤트와 이벤트 데이터는 각각 오퍼레이션의 이름과 매개변수로 변환된다.

변환 규칙 10:동적 모델의 이벤트 데이터 중 ObjectID는 오퍼레이션의 선 조건문에서 해당 인스턴스가 실제로 존재하는지 테스트되는 문장으로 변환된다.

변환 규칙 11:동적 모델의 상태는 오퍼레이션의 조건 영역에 해당되며, 전이가 일어나기 전의 상태는 선 조건문, 전이가 일어난 후의 상태는 후 조건문에서 테스트되는 문장으로 변환된다.



(그림 1) 동적 모델에서의 상태 전이 표현의 기본 형태
(Fig. 1) Basic part of state transition in dynamic model

(그림 1)과 같은 형태의 동적 모델로부터 기본적인 VDM 오퍼레이션 정의역을 다음과 같이 변환한다.

Name(ID:ObjectRecordName, ...)
pre:ID ∈ ObjectRecord ∧ status = '상태 1'
post:status = '상태 2'

동적 모델은 정보 모델의 각 객체에 대해 구축하며 각 이벤트에서 ObjectID를 통해 어떤 인스탄스에게 보내질 이벤트인지 명시하게 된다. 오퍼레이션 명세

에서 선행 조건의 강화는 오퍼레이션 적용 범위를 제한하는 것이 되므로 변형될 VDM 명세의 선행조건은 이벤트 수신 인스탄스에 대한 존재 유무를 나타내는 존재 제약성과 이벤트 관련 자료에 관한 제약 조건, 그리고 이벤트를 수신 가능한 상태인지 확인하는 것으로만 정의한다.

변환 규칙 12:동적 모델의 행동은 오퍼레이션의 후조건문에서 행동의 수행 여부를 테스트하는 문장으로 변환되며 행위가 여러 개일 경우에는 논리 연산자 "and"를 이용하여 조건에 추가되어 변환된다.

변환 규칙 13:동적 모델의 행위 중에서 매개 변수 외의 외부 자료에 대한 수정이 있을 때 오퍼레이션의 변수 선언에서 ext wr 타입의 변수로 변환되며, 그 외의 경우는 ext rd 타입의 변수로 변환된다.

행위의 분석을 통해 상태 정의역 중 외부 상태 요소로 정의해야 하는 부분을 추출할 수 있다. 이 외부 상태 요소는 행위에서 사용되는 속성들의 분석으로 ext rd, ext wr의 특성을 추출한다. 이벤트를 수신한 객체의 속성의 판독 및 갱신은 외부 상태 정의역으로 간주하지 않고, 다른 객체의 속성을 판독하거나 갱신할 때 그 속성을 포함한 객체의 VDM 상태 정의역을 외부 요소로 간주한다.

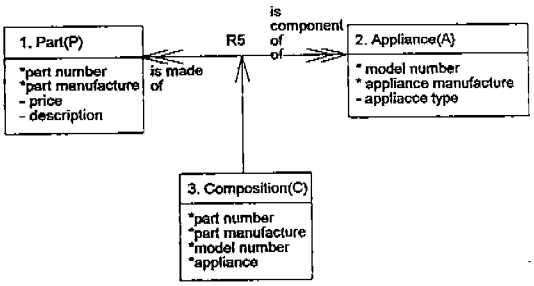
변환 규칙 14:동적 모델의 행위 중 메세지 전송문이 있을 경우, 오퍼레이션의 후조건문에서 메세지 전송문에 해당하는 오퍼레이션의 후조건문이 추가되어 변환된다.

변환 규칙 15:행위중 다른 객체의 존재 유무를 확인하는 문장은 존재 한정자(existential qualifier)를 가지는 문장으로 변환되어 조건문에 추가된다.

이때, VDM 상태 정의역에 1:M 또는 M:M의 관련성에 대한 상태 정의역이 정의되지 않았다면 정보 모델의 모순을 발견한 것이다. 즉 실제 관련성은 1:1의 관계이나 이는 해당 관련성이 성립하는 시점에서의 요소 수이고, 또 다른 1:M의 관련성이 존재하거나 1:1의 관계를 성립하기 위해 경쟁이 존재하는 경우로 그 경쟁을 해결하는 할당자 객체가 누락된 경우로 정보 모델에 새로운 객체를 추가한다.

3.3 변환 규칙을 적용한 객체 모델의 명세화 예

다음의 객체 모델은 Associative Object인 Composition이 모델링 된 부품과 제품간의 객체 모델이다. 이를 변형한 VDM은 다음 (그림 3)과 같다.

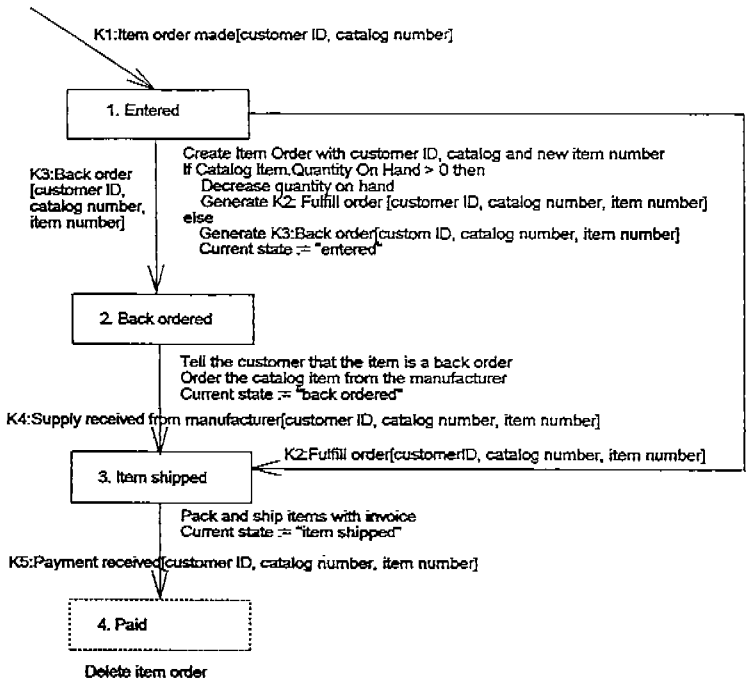


(그림 2) 정보 모델 예
(Fig. 2) Example of information model

(그림 2)의 Part, Appliance, Composition 객체의 경우는 두개의 식별자를 갖는데 이는 (변환 규칙 2)에 따라서 PartID의 특별한 식별자 정의역을 구성한다. Part와 Appliance 사이의 관련성의 Map 타입으로 변환되었으며, 다중성의 관계가 다대다 이고 Composition의 관련 객체가 모델링 되어 있으므로 별도의 상태 정의역으로 변환한다. 변환한 Map 타입의 경우는 치역의 Part나 Appliance가 여러 값일 수 있으므로 (변환 규칙 5)에 의해 새로운 Map 타입의 치역을 Set

```
PartID :: Part number
          Part manufcture
Part :: ID : PartID
        price
        description
Part-Set : set of Part
Appliance-Set   : set of Appliance
Composition :: IsMadeOf : Appliance → Part-Set
               IsComponentOf : Part → Appliance-Set
inv mk-Part-Set(made : IsMadeOf) △
    ∀ part_i, part_j ∈ rng(made)
    dom (made ▷ part_i) = dom (made ▷ part_j)
inv mk-Appliance-Set(Component : IsComponentOf) △
    ∀ appl_i, appl_j ∈ rng(Component)
    dom (Component ▷ appl_i) = dom (Component ▷ appl_j)
```

(그림 3) 변환한 결과의 일부
(Fig. 3) Partial result of transformation from (Fig. 2)



(그림 4) Item Order에 대한 동적 모델
(Fig 4) Dynamic model of object Item Order

타입의 도입으로 해결 가능하다. (변환 규칙 3)에 의해 맵 타입에 대한 Invariant를 작성하였다.

Customer ID, Catalog Number, item number, current state의 네 가지 속성을 정의하고 있는 Item 객체의 동적 모델은 (그림 4)와 같다.

이 모델에 대한 변형 결과는 다음과 같다.

```
ItemOrderMade(c:Customer, cn:Catalog Number,
              i:ItemOrder)
ext rd CatalogItem:ItemofCatalog
pre :i∈ItemOrder-set
post:i∈ItemOrder-set and i.CustomerID
       =c.CustomerID and
     i.CatalogNumber=c.CtalogNumber and
       i.itemNumber=rand()
     and if CatalogItem. QuantityOnhnd>0
       then post-FulfullOrder(c, cu, i.itemNumber)
       else post-BackOrder(c, cu, i.itemNumber)
     and i.Status="Entered"
```

3.4 규칙의 정확성 검증

본 논문에서는 객체 모델의 시뮬레이션을 위한 내부 표현 명세를 위해 변형 규칙의 정의를 통해 형식 명세 언어인 VDM을 이용한 명세를 생성하였다. 변환된 VDM 명세의 정확성을 보장하기 위해서는 정의한 변환 규칙의 검증이 필요하다. 변환 규칙의 검증을 위해서는 정확성(Correctness), 완전성(Completeness)을 확인해야 한다[13].

3.4.1 정확성

정확성을 확인하기 위해서는 변환 이전의 객체 모델이 갖고 있는 정보와 변환 규칙의 적용으로 생성한 VDM 명세가 표현하는 모델 사이의 정보 손실 또는 추가가 없음을 보장해야 한다. 그러므로 자료 정의역과 행위 정의역을 구분하여 정보의 침삭 여부를 확인한다. 자료적 측면의 검증은 데이터 값들의 보존 여부로 확인할 수 있다. (그림 2)의 객체 모델은 (그림 5)의 정보를 구성하게 된다[14].

객체 모델을 변형하여 얻어진 (그림 3)의 VDM의

Part Number	Part Manufacturer	Price	Description
103	GE	$100	
21	Seara	$523	Not in stock

(a) Part

Model Number	Appliance	Appliance
176A	Washer	GE
92AB	Dryer	Maytag

(b) Appliance

Part Number	Part Manufacturer	Model Number	Appliance Manufacturer
103	GE	176A	GE
103	GE	92AB	Maytag
21	Seara	92AB	Sears

(c) Composition

(그림 5) (그림 2)의 정보 모델이 표현하는 정보 예
(Fig. 5) Information from (Fig. 2)

Customer ID	Catalog Number	Item Number	Current State
240	RES123	234	Entered
124	BOOK14	111	Entered
688	CD789	567	Back Ordered
135	RECORD12	135	Item Shiped

(a) K1:ItemOrederMade(100,120) 이벤트 발생 이전의 객체 상태 테이블

<단, CatalogItem.QuantityOnHand 의 상태가 0 이상으로 가정>

Customer ID	Catalog Number	Item Number	Current State
240	RES123	234	Entered
124	BOOK14	111	Entered
688	CD789	567	Back Ordered
135	RECORD12	135	Item Shipped
100	120	145	Entered

(b) K1:ItemOrederMade(100,120) 이벤트 발생 직후의 객체 상태 테이블

Customer ID	Catalog Number	Item Number	Current State
240	RES123	234	Entered
124	BOOK14	111	Entered
688	CD789	567	Back Ordered
135	RECORD12	135	Item Shipped
100	120	145	Item Shipped

(c) K1:ItemOrederMade(100,120) 이벤트 처리 과정에서 발생한
K2 이벤트에 의한 객체 상태 테이블

(그림 6) 객체 Item Order의 상태 변화 과정
(Fig. 6) Stage of transition of Item Order

상태 정의역에서 얻어지는 정보는 다음과 같다.

Part={(103, "GE", $100, " "), (21, "Seara", $523, "Not in Stock")}

Appliance={(176A, "GE", "Washer"), (92AB, "Maytag", "Dryer")}

Composition={{(103, "GE")→{(176A, "GE"), (92AB, "Maytag")}}, {(21, "Seara")→{(92AB, "Seara")}}, {(176A, "GE")→{(103, "GE")}}, {(92AB, "Maytag")→{(103, "GE")}}}

변형된 VDM에서 얻어진 정보는 변형 이전의 객체 모델이 갖는 정보와 동일함을 알 수 있다. 이로써 정보 모델의 변형 규칙의 정확성을 검증할 수 있다.

ItemOrder와 연관된 객체 Catalog의 상태를 가정한 단계를 거치기는 하였으나 이는 현재 동작 중인 다른 객체의 상태 테이블의 파악으로 간주할 수 있으므로, 객체의 상태 변화 테이블을 (그림 6)으로 간주할 수 있다. 이를 변환하여 만든 VDM의 오퍼레이션 명세 중 일부는 다음과 같다. 변환한 VDM명세의 일부 오퍼레이션 명세를 기반으로 상태 정의역을 확인하면 (그림 6)과 동일한 결과를 얻게 되어 동적 모델의 변형 규칙 역시 정확함을 확인할 수 있다.

3.4.2 완전성

완전성은 객체 모델의 요소 전부가 VDM의 구성요소로 변환됨을 검증하는 것이고 OOSA의 정적 모델과 동적 모델의 각 구성요소와 VDM간의 대응성은 〈표 1〉과 같이 정리 할 수 있다.

〈표 1〉 객체 모델과 VDM간의 대응 관계
〈Table 1〉 Correspondence between object model and transformed VDM

객체 모델		VDM	객체 모델		VDM
정보모델	객체	Record Type, Set Type	동적모델	상태	Pre 또는 Post
	속성	Fields of Record		이벤트	오퍼레이션 또는 함수
	관련성	Map Type		전이	Pre와 Post의 명세로 상태 변화 표현
	상속성	Record의 상속		행동	Post를 구성하는 논리 연산문을 And로 조합

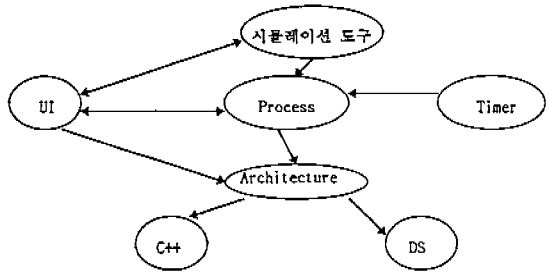
〈표 1〉에서 보듯이 객체 모델의 모든 정보를 VDM으로 대응하고 있으므로 완전성을 보장한다.

4. 시뮬레이션에 의한 객체 모델의 확인

프로토타입은 향후 개발하고자 하는 시스템의 본질적 특성만을 보여주는 개발 초기의 모델이다. 본 논문에서는 형식 방법론을 기반으로 객체 지향 분석 모델을 명세하게 하고, 이를 바탕으로 모델과 사용자의 요구간의 확인을 위한 시뮬레이션 환경을 구축하였다. 분석 모델에 명세된 객체와 그들의 속성 및 오퍼레이션과 동적 모델에 분석한 동적 특성으로 사용자의 기능적 요구 사항을 만족할 수 있는지 확인하고자 하는 것이다. 시뮬레이션을 통한 프로토타입 구축으로 확인 단계를 지원하는 시뮬레이터를 개발하였다.

4.1 시뮬레이션 도구의 설계

개발한 시뮬레이션 도구는 OOSA방법에 의한 객체 모델링의 결과물인 정보 모델과 동적 모델을 참조하여 각 객체와 시스템 내의 상호 작용을 동적인 시각 모형으로 보여주는 것이다. 시뮬레이션 개발의 첫단계로 시뮬레이션 도구의 도메인을 크게 객체 모델 편집기, 뷰어, 시뮬레이션 처리기로 분할하였다. 객체 모델 편집기는 시뮬레이션 도구의 입력 정보를 OOSA 방법론에서 지원하는 다이어그램을 이용하여 생성한다. 뷰어는 현재 진행 중인 시뮬레이션의 상황을 보여준다. 시뮬레이션 처리기는 시뮬레이션에 필요한 정보를 객체 모델 편집기를 통하여 입력받아 시뮬레이션을 수행하여 뷰어를 통하여 사용자에게 보여준다. 이를 OOSA 방법론의 도메인 차트로 표현하면 다음과 같다.



(그림 7) 시뮬레이션 도구에 대한 도메인 차트
(Fig 7) Domain chart of simulation tool

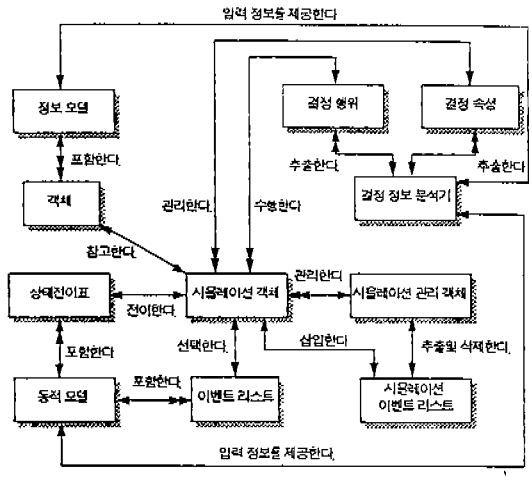
각 도메인과 도메인간의 연결성은 〈표 2〉과 같다.

이들 도메인 중 시뮬레이션 도구의 가장 중요한 핵심 도메인인 시뮬레이션 도구를 3개의 서브 시스템,

〈표 2〉 도메인과 도메인과의 연결성
〈Table 2〉 Description of domain and bridge

도메인 및 연결성	설 명
시뮬레이션 도구	부석 모델로부터 시뮬레이션 정보를 추출하여 시뮬레이션 과정을 제어
사용자 인터페이스(UI)	시뮬레이션의 결과를 처리하며 사용자의 부가 정보 입력을 처리
프로세스(Process)	시뮬레이션 과정에 생성하게 되는 각 객체의 처리 행위
타이머(Timer)	동적 모델에 내포된 객체로 시뮬레이션에 필요한 시간적 제약성 처리
구조 (Architecture)	설계에 필요한 객체들의 집합으로 언어, 각종 자료 구조 객체들의 집합
시뮬레이션 도구 ⇒UI	시뮬레이션 과정 중에 생성한 객체의 상태에 관한 정보를 제공
UI⇒ 시뮬레이션 도구	시뮬레이션에 필요한 부가 정보를 사용자로부터 입력받아 제공
시뮬레이션 도구 ⇒프로세스	객체의 상호 작용 및 상태 전의 처리 결과 제공
타이머 ⇒프로세스	프로세스 수행 또는 지연 시간 제약

객체 모델 편집기, 뷰어, 시뮬레이션 처리기로 분할하여 설계 하였다. 객체 모델 편집기는 사용자가 쉽게 객체 모델을 작성할 수 있도록 다이어그램을 제공한다. 편집기에서 제공하는 다이어그램은 OOSA의 정보 모델에서 필요로 하는 요소와 동적 모델에서 필요로 하는 요소로 구성된다. 정보 모델에서 필요한 요소는 객체, 관련성, is-a 관련성이며, 동적 모델에서 필요한 요소는 상태, 상태 전이, 이벤트, 행동 등이다. 이러한 요소를 제공하여 사용자가 쉽게 모델링할 수 있는 기능을 제공하며 모델 자체를 시뮬레이션 도구의 입력으로 사용할 수 있도록 3장에서 정의한 모델 명세 언어로 저장한다. 뷰어는 시뮬레이션 진행 상황을 보여주는 것으로 한 화면에 3개의 윈도우로 분할하여 시뮬레이션 진행 사항광 인스탄스의 사태, 처리 행동등의 다양한 관점에서의 뷰를 제공한다. 시뮬레이션을 관리하고 사용자와의 상호 작용을 처리하는 시뮬레이션 도구의 가장 중요한 부분이다. 이벤트 리스트를 통하여 시뮬레이션의 스케줄을 결정하고, 실제로 이벤트가 도달할 객체에게 메시지를 전달하여 전이를 일으켜 실제 시뮬레이션을 담당하여 그 결과를 뷰어를 통하여 보여준다. 위의 3가지 서브 시스템을 분석하여 객체 모델링한 결과는 (그림 8)과 같다.



(그림 8) 시뮬레이션 도구의 정보 모델
(Fig 8) Information model of simulation tool

4.1.1 입력 정보 분석기

시뮬레이션의 진행은 OOSA의 동적 모델을 기반으로 이루어진다. 입력 정보 분석기의 역할은 다음과 같다.

①정보 모델에 나타난 객체를 몇 개 인스탄스화 시킬 것인지 결정한다. 정보 모델에서 항상 1:1의 관련성만 갖는다면 시뮬레이션 상황에서도 1개의 인스탄스만 필요하나, 1:M의 관련성에서 M쪽에 해당하는 객체라면 여러 개의 객체가 인스탄스화 될 수 있기 때문이다.

②각 상태마다 수행해야 하는 행위 중 논리 연산의 대상이 되는 결정 속성을 찾는다.

③결정 속성의 값을 변화시키는 결정 행위를 찾는다.

입력 정보 분석기의 속성들은 시뮬레이션이 시작되기 전에 시뮬레이션 객체를 초기화하기 위하여 사용된다.

4.1.2 시뮬레이션 객체

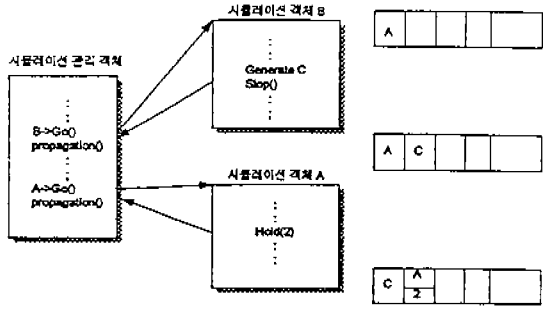
시뮬레이션 객체는 OOSA 방법론에 의해 만들어진 각각의 객체에 시뮬레이션의 진행에 필요한 기능이 추가되어 만들어진 클래스이다. 시뮬레이션 객체에서 사용되는 OOSA의 정보 모델과 동적 모델의 정보

는 결정 속성 리스트, 결정 행위 리스트, 상태 전이표, 이벤트 리스트이며 그 외에 현재 상태 변수가 추가된다. 시뮬레이션을 위해서 추가된 기능은 〈표 3〉과 같다.

〈표 3〉 시뮬레이션을 위해 추가된 인터페이스 오퍼레이션
〈Table 3〉 Additional operation needed for simulation

Go( )	객체에 도달한 이벤트에 대하여 상태 전이를 하며 새로운 상태에서 처리해야 하는 결정 행위를 수행
Stop( )	현재 수행 중인 시뮬레이션 객체의 수행을 보류
Hold(T)	주어진 시간 T동안 시뮬레이션 객체의 수행을 보류
Time( )	현재 시간을 알림
Create( )	시뮬레이션 객체의 인스탄스화
Delete( )	시뮬레이션 객체의 삭제

위에서 정의한 시뮬레이션 기능은 주로 시뮬레이션 프로세스의 동기화를 위해서 필요하다. (그림 9)에서 시뮬레이션을 진행하는 인스탄스는 B와 A가 있다. 시뮬레이션 관리 객체에서 이벤트 리스트의 시작 노드인 B에게 전달되는 이벤트를 처리하기 위해서 B→Go를 호출한다. 이 순간 제어의 흐름이 B로 이동하여 해당 이벤트에 대한 상태 전이를 하고, 새로운 상태에서의 결정 행위를 수행한다. 그리고 Stop( ) 함수를 호출하면 제어의 흐름은 다시 시뮬레이션 관리 객체로 이동한다. 시뮬레이션 관리 객체는 다시 이벤트 리스트의 시작 노드에 해당하는 이벤트를 위와 같은 과정을 거쳐 처리한다. 만약 결정 행위의 수행 중에 새로운 이벤트를 생성하는 행위가 있다면 생성된 이벤트는 광역 이벤트 리스트의 마지막에 삽입된다.



(그림 9) 시뮬레이션 프로세스의 동기화
(Fig 9) Process synchronization of simulation

시뮬레이션 객체의 속성은 입력 정보 분석기에 의해 초기화되며 객체의 인스탄스 갯수와 결정 속성의 값은 시뮬레이션이 시작되기 전에 사용자에게 입력받아야 한다.

4.1.3 시뮬레이션 관리 객체

시뮬레이션 관리 객체는 (그림 9)에서 나타난 바와 같이 광역 이벤트 리스트를 통하여 시뮬레이션 객체를 관리하여 전체 시뮬레이션을 진행시킨다. 시뮬레이션 관리 객체는 다음의 세가지 중요한 역할을 수행한다.

①시뮬레이션 객체의 스케쥴링을 담당한다. 광역 이벤트 리스트를 통하여 다음에 수행할 시뮬레이션 객체를 결정하여 시뮬레이션 객체를 스케쥴링 한다.

②시뮬레이션 제어의 흐름을 관리한다. 시뮬레이션 객체의 활성화 함수를 호출하여 제어의 흐름을 해당 객체로 넘기고 비활성화 함수가 호출될 때 다시 제어의 흐름을 되넘겨 받는다.

```
class Sim_Obj
{ private :        int InstanceID;                    // 시뮬레이션 객체의 인스탄스 이름
        char object_name[10];              // 시뮬레이션 객체의 정보 모델에서의 이름
        int num_state;                                 // 상태의 수
        Determinent_Attr_List *attrlist;   // 결정 속성 리스트
        STT stt;                                       // 상태 전이표
        ACTION_HANDLER_List *actionlist;               // 결정 행위 리스트
        int current_state;                             // 현재 상태 속성
        Event_List *eventlist;             // 이벤트 리스트
```

```
public :            Sim_Obj();
      ~Sim_Obj();void Go(int e);
      void Stop();              void Hold(Time t);
      void Create();            void Delete();
      void Set_InstanceID(int i);
      void Set_object_name(char *s);
      void Set_no_state(int i);
      void Set_determinent_attr(DETERMINENT_ATTR *head);
      void Set_Event_List(Event_List *e);
      void Set_stt(STT s, int size);
      void Set_cur_state();
      int  Get_DAttr_Value(char *a);
      int  Get_cur_st();
      int  Select_Event();
};
class Sim_Mgt
{ private :
       Sim_Obj_List      objlist;
       int               objnum;
  public:
      Sim_Mgt();
      ~Sim_Mgt();
      void Manage();
      void Init();
      void Propagation();
      Sim_Obj *Select_Instance();
      void ShowTimer(int);}
```

(그림 10) 시뮬레이션 클래스와 관리 클래스
(Fig. 10) Description of simulation class and management class

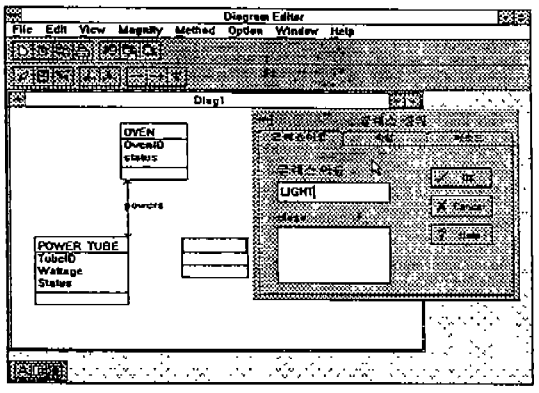
③변경된 결정 속성의 값을 모든 시뮬레이션 객체에게 반영한다. 결정 속성은 객체 내부의 속성일 수도 있고, 외부 객체의 속성일 수도 있다. 따라서 한 객체 내의 결정 속성이 변경되면 이를 결정 속성으로 가지는 다른 모든 객체의 값들도 변화시켜 일치성을 유지해야 한다.

앞에서 이루어진 시뮬레이션 도구에 대한 설계의 결과를 바탕으로 시뮬레이션 도구의 구성요소인 시뮬레이션 객체와 관리 객체의 상세 설계 모델은 (그림 10)이다. 시뮬레이션 객체는 OOSA의 산출물과 결정 정보를 속성으로 가지며, 오퍼레이션은 이러한 속성을 관리하는 오퍼레이션과 시뮬레이션 기능을 위한 오퍼레이션으로 구성된다.

4.2 시뮬레이션 도구의 구현

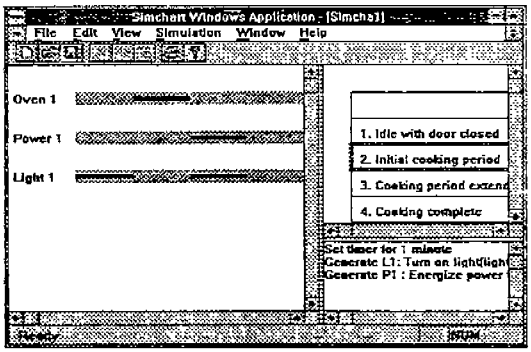
시뮬레이션 도구는 PC Windows 95환경에서 Visual C++ 4.0을 사용하여 개발되었다.

(그림 11)는 Oven, Power Tube, 그리고 Light로 구



(그림 11) 객체 모델 편집기의 실행 예
(Fig. 11) Snapshot of object model editor

성된 정보 모델을 객체 모델 편집기를 이용하여 그리는 화면이다. (그림 12)은 시뮬레이션 도구의 실행 화면으로 Oven, Power Tube, Light의 객체를 각각 1개씩 인스탄스화하여 시뮬레이션 한다. 그림에서 가로축은 이벤트의 처리 순서를 의미한다. 먼저 Light 객체에 이벤트가 도달하고 그리고 Oven 객체, 다음은 Power Tube와 Light 객체에 동시에 이벤트가 도달함을 의미한다.



(그림 12) 시뮬레이션 화면
(Fig. 12) Screen for simulation step

4.3 평 가

객체 모델의 사용자 확인을 위하여 프로토타입을 생성하는 방법은 프로토타이핑 언어를 사용하는 방법과 실행 가능한 형식 명세를 사용하는 방법, Embley가 제안한 CASE인 IPOST를 사용하는 방법이 있다. 본 논문에서는 위의 3가지 방법과 본 논문에서 제안한 방법을 다음의 기준으로 비교 평가 하였다.

4.3.1 평가 요소 정의

본 논문에서 구축한 확인 지원 시스템에 관한 평가를 위해서 다음의 네 가지 평가 기준을 정립하였다.

1)모델 구축 능력 관점:프로토타이핑은 전체 소프트웨어 시스템을 전통적인 생명주기에 따라 개발하는 것에 비하여 그 목적과 방법에 차이가 있으나 소프트웨어를 개발한다는 시각에서 보면 크게 다를 바가 없다. 프로토타입을 통해 시스템의 각 부분을 하나의 increment로 개발할 수 있다면 각 increment를 개발하는 데 필요한 효과적인 모델을 제시해야 한다.

2)모델의 변환 필요성:프로토타이핑을 위해 모델 자체를 다시 도구나 언어에 맞게 재구성을 위한 변환이 필요할 수도 있다. 이러한 변환이 필요한 경우는 변환에 필요한 검증된 규칙과 이를 지원하는 도구가

〈표 4〉 평가 결과
〈Table 4〉 Evaluation table

	프로토타이핑언어	실행가능한 형식 명세	IPOST	본 논문
모델 구축 환경		-명세 자체가 모델을 표현	-OSA 방법론 지원 -방법론의 적용범위가 한정적	-S/M의 OOSA 방법론 지원 -널리 사용되는 방법론 지원 -타방법론으로의 확장 용이
모델 변환 필요성	-변환 필요 -수작업에 의한 변환	-필요하지 않음 -논리 언어 사용 -명세 자체가 실행가능함	-필요하지 않음 -도구 자체의 기본 모델 사용	-필요하지 않음 -분석 단계의 결과물인 객체 모델이 직접 적용
이해 용이성	-프로그래밍 언어 습득 필요	-수학적인 기호사용 -이해하기 어려움	-다이어그램이외의 형식 명세 기법의 사용으로 사용자의 이해도 저하	-다이어그램 형태로 제공 -사용자 관점이 일관성을 위해 명세 도구로 다이어그램만 사용
시각화	-언어에 따라 다양함 단순한 사용자 인터페이스만 제공하는 것이 일반적		-지원 -UI는 개념적인 인터페이스로 제공 -객체의 상태 네크워, 관련성	-다양한 관점에서 지원 -3개의 뷰 제공 시뮬레이션 진행윈도우, 상태 머신 윈도우, 행위 윈도우

필요하므로, 부가적인 개발 비용이 든다.

3) 이해의 용이성 : 프로토타입의 대상이 되는 모델의 사용자 친숙도를 평가하는 것이다.

4) 시각화 : 프로토타이핑 과정의 시각화를 통해 시스템 수행의 일반적 과정을 추적할 수 있는 기능성 제공의 평가이다.

4.3.2 평가 결과

이들 관점에서의 평가 결과는 〈표 4-3〉과 같다.

〈표 4〉에서 보는 바와 같이, 개발 과정의 단계의 산물인 객체 모델의 사용으로 개발자와 고객간의 일관된 인터페이스를 유지로 이해도 증진의 장점을 갖는다. 내부 형식 명세 모델을 바탕으로 검증이 가능하고 뿐만 아니라, 이의 시각화로 사용자 확인을 지원한다는 장점도 갖는다.

5. 결 론

소프트웨어 개발에서 초기 과정인 분석과 설계 단계에서의 오류의 검출은 전체 소프트웨어 개발 노력과 유지 보수 비용을 감소시킬 수 있다. 객체 모델링과 형식 명세 기법은 이러한 개발 초기 단계의 부정확한 산출물과 불충분한 모델 구축의 문제를 해결할 수 있는 방법론이다. 객체 모델링은 고객의 요구 사항 추출, 표현의 편리함, 이해 용이성과 같은 장점이 있지만, 비 정형화된 방법으로 인하여 모호성과 비정확성을 내포하고 있다. 그러고 형식 명세 기법은 명세에 대한 정확성, 명확성, 간결성을 보장하지만 명세 자체가 수학 이론을 기반으로 하기 때문에 많은 비용을 요구한다.

본 논문에서는 변환 기법을 적용하여 상호 보완적인 두 방법론의 장점들을 최대화하였다. 이를 위하여 본 논문에서는 객체 모델을 형식 명세 언어인 VDM으로 자동 변환시키기 위한 변환 규칙을 확립하여 변환한 VDM 명세를 기반으로 시뮬레이션을 통한 프로토타이핑 단계에 적용하였다.

변환 방법은 Shlaer/Mellor의 정보 모델에서 표현되는 객체와 관련성을 VDM에서 제공하는 6개의 데이터 구조를 이용하여 VDM의 상태 영역으로 변환하였고, 동적 모델의 상태, 이벤트, 이벤트 데이터, 행위를 VDM의 오퍼레이션 영역으로 변환하였다. 이를 통해 소프트웨어 개발단계인 초기 과정에서 분석가에게 모델링의 편리함과 자율적인 재량을 제공함과 동시에 모델에 대한 정형적인 검증 방법을 제공한다는 장점을 얻을 수 있다. 그리고 비형식적인 객체 모델을 형식 명세 언어인 VDM으로 변환하므로써 객체 모델을 정형화된 방법으로 검증할 수 있는 수단을 제공한다.

이와 함께 고객의 요구 사항에 대한 확인 작업은 이미 검증된 객체 모델의 시뮬레이션을 통한 동적 시각 모형인 프로토타입으로 지원하였다. 시뮬레이션은 객체 모델링의 결과물인 정보 모델과 동적 모델을 참조하여 시스템 내의 각 객체간의 상호 작용을 동적인 시각 모형으로 제공하여, 개발자들과 사용자들간의 의사 소통상의 효과를 크게 증진시킬 수 있다.

앞으로의 과제는 변환된 VDM 명세를 검증 과정에서 수정할 경우, 객체 모델의 수정이 수작업이 아닌 자동으로 이루어질 수 있는 역 공학 도구에 대한 연구와 실시간 시스템의 시간 제약성을 나타내기 위한 VDM 명세의 확장 방안에 대한 연구가 필요하다.

참 고 문 헌

[1] 이경환, '소프트웨어 재사용을 위한 객체 모델링 기법', 교학사, 1993.

[2] 이주헌, '소프트웨어 생산공학론', 법영사, 1993.

[3] David W. Embly, Barry D. Kurtz, Scott N. Woodfield, 'Object-Oriented Systems Analysis : A Model-Driven Approach', Yourdon Press, 1992.

[4] Grady Booch, 'Object-Oriented Analysis and Design', The Bemjamin/Cummings Publishing Company

[5] Sally Shlaer and Stephen J. Mellor, 'Object-Oriented System Anaysis : Object Life Cycles Modeling the World in States', Prentice Hall, 1992.

[6] Bob Fields, "A guide to reading VDM specification," Technical Report UMCS-92-12-4, Department of Computer Science, University of Manchester, 1992.

[7] Jonathan P. Bowen, Michael G. Hinchey, "Ten Commandments of Formal Methods", IEEE Computer, Vol. 28, No. 4, April 1995.

[8] Kanth Miriyala, Mehdi T. Harandi, "Automatic

Derivation of Formal Software Specifications From Informal Descriptions," IEEE Transactions on Software Engineering Vol. 17, No. 10, 1991.

[9] Martin D. Fraser, Kuldeep Kumar, Vijay K. Vaishnavi, "Informal and Formal Requirements Specification Languages: Bridging the Gap," IEEE Transactions on Software Engineering Vol. 17, No. 5, May, pp. 454-466, 1991.

[10] Nico Plat, Peter Gorm Larsen, "An Overview of the ISO/VDM-SL Standard," ACM SIGPLAN Notices, Vol. 17, No 8, 1992.

[11] Norbert E. Fuchs, "Specification Are (Preferably) Executable," Software Engineering Journal, 1992.

[12] Robert B. Jackson, David W. Embley, Scott N. Woodfield, "Automated Support for the Development of Formal Object-Oriented Requrements Specification," Proceedings of 6th International Conference on CAiSE'94, 1994.

[13] Richard A. Kemmer, "Integrating Formal Methods into the Development Process," IEEE Computer Vol. 23, No 10, 1990.

[14] Paul L. Bergstein, "Object-Preserving Class Transformation," SIGPLAN Notices, Vol. 26, No. 11, pp. 299-313, 1991.

[15] Sally Shlaer and Stephen J. Mellor, 'Object-Oriented System Anaysis: Modeling the World in Data', Prentice-Hall, 1988.

정 란

1976년 중앙대학교 전자계산학과 학사
1983년 중앙대학교 대학원 전자계산학과 석사
1993년~1996년 대구효성카톨릭대학교 전산통계학과 박사과정 수료
1983년~현재 삼척산업대학교 전자계산학과 교수
관심분야: 소프트웨어공학, 지식공학

김 정 아

1988년 중앙대학교 전산과 졸업(이학사)
1990년 중앙대학교 대학원 전자계산학과 졸업(공학석사)
1994년 중앙대학교 대학원 전자계산학과 졸업(공학박사)
1994년~1996년 2월 중앙대학교 Post-Doc.
1996년 2월~현재 관동대학교 컴퓨터 교육과 조교수
관심분야: 소프트웨어 공학, 재사용 이론, 형식 명세 기법, 객체지향 개발 방법론

문 충 렬

1994년 중앙대학교 전자계산학과 졸업(공학학사)
1995년~현재 중앙대학교 대학원 컴퓨터공학과 재학중(석사과정)
관심분야: 객체지향 형식 명세 방법, 재사용, 객체 모델의 검증

김 정 두

1987년 영남대학교 공학박사
1966년~1967년 미국 국무성 초청 TDP 참가
1981년~1982년 미국 Southern Illinous 대학 연수
1983년~현재 대구효성카톨릭대학교 전자정보공학부 교수
관심분야: 지식공학, 학습시스템 개발