객체지향기법에서 Static 모델과 Event와의 통합 방안 연구

*박 회 일 *맹 경 재 **김 정 태

* 한국외국어대학교 대학원 경영정보학과

** 한국외국어대학교 경영정보학과 교수

Abstract

객체지향 기술에서 개념적 모델링을 위해 크게 객체의 정적인 측면과 동적인 측면의 두가지 관점으로 접근하고 있다. 본 연구에서는 시스템의 파악과 구현을 위해 이러한 두가지 이질적인 관점을 나누어 접근하는 것이 아니라 객체의 정적인 측면(Static)과 동적인 측면인 Event를 통합한 방법론을 제시하고 있다.

서 론

객체지향 기술은 소프트웨어 개발에 생산적인 방법과 실제를 제공하며, 가장 넓은 응용분야에 적용되고 있다[14]. 객체지향 기술은 노르웨이의 K.Nygard가 Simula 67 프로그래밍 언어를 개발하면서 객체지향의 개념을 사용한 이래 지난 몇 년간 큰 발전을 거듭해 왔다. 객체지향 기술에 대한 연구영역을 보면 크게 두가지 영역으로 구분할 수 있는데 하나는 객체지향 설계방법론에 관한 연구이며 다른 하나는 실제 객체지향 기술을 구현 혹은 응용하는 분야이다. 본고에서는 이중 객체지향 설계방법론에 대하여 논의하고자 한다.

객체지향 방법론 간의 특성에 대한 고찰은 향상된 방법론의 개발에 도움을 줄 것으로 기대된다. 객체지향 방법은 분석과 설계의 두 영역으로 구분될 수 있는데 분석에서는 객체를 구조화하며, 설계에서는 객체의 행위를 기술하며 실제 구현에 집목시키게 된다. 객체지향 분석 및 설계에서 객체의 구조화와 행위의 표현은 방법론의 저자들 마다 서로 다른 표현기법을 이용하고 있으며, 객체 구조에 대한 연구들은 비교적 일반적인 개념으로 정착되고 있지만 행위에 대한 표현은 아직 일반성을 갖지 못한 상태다[9]. 객체 행위에 대한 표현을 기호화 하는것은 최종사용자 인터페이스의 구현과 소프트웨어 공학에서 문서화에 유용하게 사용된다[11,13].

본 연구에서는 이러한 객체지향 기술에서 구조화 영역에 행위를 표현하기 위해 기존의 방법론을 간략하게 기술하며, 객체 행위의 표현을 비교 분석하기 위해 정보공학측면을 포함하고 있는 Martin의 OOM(Object-Orientde Method), 객체지향 기술에서 가장 보편화되 인용되는 Rumbaugh의 OMT (Object-Oriented Modeling Technique), 구조화 영역과 행위 영역의 구별에 비교적 명확한 표현을 가진 Desfray의 OE(Object Engineering), 행위영역에서 시간 개념의 표현이 풍부한 Embley의 OSA (Object-Oriented Systems Analysis), 끝으로 객체지향 설계시 지원이 가장 풍부한 Booch의 OOD (Object-Oriented Design)의 방법론을 행위 중심으로 분석한 후, 보다 향상된 행위중심의 객체지향 방법론에 대한 개념적 틀을 제시하고자 한다. 이들 행위의 표현들이 실제 분석과 설계 그리고 구현에 있어 일관성을 유지하도록 하는데 본 연구에 의의가 있다.

1. 객체지향 분석의 기본개념

객체지향 분석의 목적은 컴퓨터시스템에 대한 사용자의 요구사항을 이해하고 기술하기 위한 것이다. 객체지향 분석방법론은 분석의 대상이되는 문제영역을 객체지향 개념을 이용하여 분석하는 기법이라고 할 수 있다. 분석단계에서의 개념적 모델은 적어도 사용자가 수행하는 업무를 지원하기 위해 시스템에 의해 제공되는 기능을 설명해야 한다.

객체지향 분석단계에서는 주어진 문제영역 내의 객체를 발견하고 그들간에 존재하는 관계나 주요 활동들을 기술해야 하는데, 이러한 주요활동을 event라 한다. Martin은 객체의 변화로 발생한 상황의 변화나 변화요구에 대해 작용하는 것을 event로 정의하며, Desfary에 의하면 event는 주목할만한 정보와 특정 시간상에서 발생하는 순간을 의미하며, Embley와 Kurtz는 event를 객체 내부와 외부에서 발생하는 하나의 개체로 간주하였다. 이들 event에 대한 공통된 관점은 객체지향 분석과정에서 시스템의 중요한 정보를 발견해 낼수 있다는 점이다. 본고에서는 몇 가지 객체지향 방법론들에서의 event 모델링에 대한 사용을 비교 분석하며, static한 모델과 event와의 통합을 위한 방법론과 그 표현 방안에 대하여 다루고 있다.

2. 기존 방법론들에 대한 고찰

2.1 OOM(Object-Oriented Method) - Martin

Martin의 OOM(Object-Oriented Method)에서는 시스템의 구조를 기술하는 object diagram과 사용자의 요구사항이나 시스템의 변화 즉 객체의 행위를 기술하기 위해 event diagram을 사용하고 있다. Event diagram에서는 operation, event, method, trigger, condition과 같은 요소를 포함하고 있다. Operation은 일반적으로 객체의 행위를 정의하기 위해 사용하는데 method, event와 함께 표현된다. Event와 operation 간의 관계에서 operation은 states의 변화를 이행하는 process를 의미하며 process의 한 단위를 포함한다. 이에 반해 event는 operation을 상세화한 표현이다. opration과 event의 관계에서 event는 하나의 객체와 하나의 operation과 관계를 가지며, 내적.외적 operation에 의해 표현되기도 한다. Operation과 event의 표현은 process를 지원하며 이것은 다시 method를 구체화 시킨다. 구체화된 method는 객체지향 실행에 있어 중요한 지침이 된다. method는operation을 위한 특별한 process로 정의되며 method와 operation과의 관계는 다음과 같다.

1. Event diagram은 계층적으로 표현 할 수 있는데 이것은 operation에 method을 표현하기 때문이다. 계층적인 operation의 method는 하위의 method를 가질 수도 있다.
2. 하나의 operation이 두 개의 다른 method를 가질 수도 있다.
3. 하나의 method가 서로 다른 객체에 사용될 수도 있다(polymorphism).

Event의 기본적인 형태는 외부적, 내부적, 그리고 임시적인 발생에 기인된다. Event의 구조는 다시 subtype과 supertype으로 나눌 수도 있으며, 기본적인 event의 형태를 시스템의 성격에 따라 분석자에 의해 특별히 구분할 수도 있다 (예 : 삭제, 갱

신, 수정, 이동 등). 임시적인 변화에 기인되는 event는 일반적으로 객체의 history를 표현 관리하는데 사용된다.

Trigger사용은 하나의 operation에서 서로 다른 object로 event가 발생할 때 trigger rule을 사용한다. Trigger의 표현이 process의 표현을 더 강력하게 해 주지만 모든 객체가 trigger을 통한 operation의 흐름을 요구하는 것은 아니다. trigger가 발생할 때 해당 operation 앞에서 제약이 필요한 경우 사용되는 control condition 표현은 지속적이고 반복적인 검색이 이루어질 때 필요하다. 일반적으로 condition은 IF와 AND, OR 등의 의미로 기술될 수 있으며 일반적으로 동기적이고 병렬적인 operation의 발생에 필요하게 된다.

2.2 OE(Object Engineering) - Desfray

Desfray의 OE(Object Engineering)는 일반적으로 객체의 구조를 정의하고 그 관계를 설정하기 위해 사용하는 structure model과 객체가 제공하는 서비스나 규칙을 표현하기 위해 사용하는 operation model, 그리고 객체를 구현하기 위한 최종 표현으로 객체의 흐름과 동적인 요소를 표현하기 위한 dynamic model로 구성되어 있다. Structure model은 다른 객체지향 방법론에서 사용하고 있는 object model과 유사하나, operation model과 dynamic model은 다른 관점을 가지고 있다. Operation model에서는 condition을 중심으로 표현하며, 한 객체가 다른 객체와 연관을 가질 때 객체 내에 존재하는 method가 변화하게 되는데, 이러한 원인은 condition에 의한 것이다.

Dynamic model은 객체지향 개념 중 객체의 캡슐화와 독립성를 보장하기 위해 시스템의 전체적 관점에서 고려된다. 시스템 내에 존재하는 객체는 서로 결합될 수 있으며, 주어진 객체가 외부의 자극이나 특별한 상황조건에 어떻게 반응하는지를 기술한다. 이러한 dynamic model에서 표현을 위해 object flow diagram, scenarios를 사용한다.

Object flow diagram에서는 다양한 data의 흐름(객체에서 method로, process에서 process로, class에서 객체로, 객체집합에서 method로 등)을 표현할 수 있다. 이 객체 흐름 표현에서 볼 수 있는 control은 looped process와 exclusion operator로 기술할 수 있으며, 객체와 method간의 복잡한 연결은 method 중심으로 단순화 시킬 수 있다. 그리고 이 객체 흐름을 상세화 하기 위해 pseudo-language를 이용하기도 한다.

Scenario diagram은 객체 흐름의 순서 등을 고려하여 나타낼 수 있다. 객체 흐름의 표현은 전체적인 객체 관계를 알 수 있지만 처리순서를 고려하지는 않는다. 따라서 scenario diagram은 input/output에 초점을 두어 표현하며, 보다 상세한 표현을 위해 sub-scenario로 재구성할 수도 있다.

Events는 클래스간의 의사소통과 클래스간에 일어나는 행위를 제어하기 위한 방법으로 사용된다. Event는 한 순간에 발생하는 신호에 대응하는 것으로, 이것은 새로운 condition의 발생 요구와 예외적인 상황에 대응하기 위한 표현이다. Event를 class수준으로 구성하여 event의 inheritance를 표현 하기도 한다. Event를 사용하게 되는 경우들은 외부의 요구가 있을 때, 비정상적인 processing이 발생할 때, 그리고 내부적인 의사소통 흐름에서 receiving이 정의되지 않은 경우에 사용된다. Event의 효율적인 사용은 Sending/Receiving중심으로 표현할 수 있지만 Sender나 Receiver 등을 표현하지 않아도 된다.

2.3 OOSA(Object-Oriented Systems Analysis) - Embley

Embely의 OOSA는 객체 구현을 Object-Relationship Model(ORM), Object-Behavior Model (OBM), Object-Interaction Model(OIM)의 3가지 모델로 사용하고 있다. ORM은 객체 구현에서 일반적으로 객체를 정의 하고 그 객체간의 관계를 정의하여 표현하는 모델이며, OBM은 시스템에서 발생하는 행위를 표현하고, OIM은 시스템에서 객체간의 상호작용이 발생하는 것을 표현하며, 하나의 클래스내에 있는 객체가 다른 클래스의 객체나 state와 상호작용 하는 것을 표현할 수 있다.

OBM은 객체의 행위를 나타낼 때 state, condition, action 등을 표현하게 되며, state는 객체의 상태, 상황, 혹은 활동 등을 표현한 것을 말한다. 객체의 state 변환 과정을 transition이라 말하는데, 이때 state를 변환시키는 condition과 event를 triggers라고 하며, action은 events를 발생시키고, objects나 relationships를 생성하거나 소멸시키고, 이를 감시하며, message를 주거나 받는다. Action은 interruptible과 noninterruptible로 나눌 수 있다.

이 모델링의 특징은 한 객체의 클래스안에서 state들을 표현하며 우측 상단에서 좌측 하단쪽으로 진행한다. 이러한 객체 클래스 박스를 state net이라고 하며, state net안에는 state뿐만아니라 event와 각종 trigger를 표현 할 수 있다. Event와 trigger을 표현하기위해 사용하는 사각형을 두 부분으로 나누어 윗 부분은 transition's trigger를 표현하며, 아래 부분은 action으로 표현한다. 구체적인 trigger의 표현은 다시 condition과 event로 나누어지는데, condition은 객체의 현상태, object의 존재여부 혹은 객체간의 relationship의 존재 여부에 대한 논리적 문장(logical statement)으로 표현된다. Event는 시스템의 어떤 변화를 표현하는데 일반적으로 출발, 멈춤, 변화 등의 형태로 표현된다. 이밖에 trigger의 구체적인 표현으로 '@'를 사용하여 when, upon 등을 표현한다. State-net의 표현에서 net안에 다중의 state와 event를 표현하며, net밖에 있는 클래스와 관계를 가지는 경우에는 점선의 화살표로 그 net안에서 표현 한다. 특수한 condition이나 event를 표현할 때 exception을 사용하는데 이는 transition상에서 세로 선으로 표시한다. 실제 action의 상자안에 제한된 시간의 범위를 "(< 30 seconds)"와 같이 표현할 수 있다. 또한 net안에 표현되는 event 사각형 위에 번호를 표현하여 그 순서를 나타낼 수 있다. 그러나 동시성의 표현을 위해서는 많은 어려운점이 있다.

2.4 OOD(Object-Oriented Design) - Booch

Booch의 OOD는 객체지향의 많은 개념을 포함하며 그 타당성을 제시하고 있다는 점에서 주목할 만하다. Booch의 OOD를 위한 표현에서 class diagram과 object diagram이 시스템의 주요 추상체를 식별하고 그 의미를 기술한다. 그러나 실제 시스템의 구현과는 밀접한 관계를 가지고 있지 않고, 단지 시스템의 논리적 관점을 표현한 것이다. Module diagram과 process diagram은 구현을 위한 소프트웨어와 하드웨어의 구성을 기술하는데 이용되므로 시스템의 물리적 관점에 해당한다. 시스템의 정적인 성질을 표현하고 있는 앞의 네가지 모델과 함께 Booch의 OOD에서는 시스템의 동적인 성질을 표현하기 위해 state transition diagram과 timing diagram을 사용하고 있다.

State transition diagram은 특정 클래스와 관련된 동적행위를 나타내기 위한 것으로 하나의 state에서 다른 state로 전환을 야기시키는 event와 state의 변화로 발생하는 action, 그리고 start state, stop state로 구성되어 있다.

Timing diagram은 object의 동적 행위를 보여주고, 동기적인 외부의 events에 대한 영향을 보여

준다. Timing diagram은 시간이 중시되는 활동을 표현하는 경우에 있어서 매우 유용하다. 그러나 모든 object diagram에 대해 timing diagrams를 표현할 필요는 없다. 이 밖에 OOD의 특징은 상세한 객체의 변화를 표현하는데 tamplate를 이용하여, 객체 개발 도구에 유용하게 사용될 수 있다.

2.5 Object-Oriented Modeling Technique(OMT) - Rumbaugh

Rumbaugh의 OMT는 object model, dynamic model, functional model로 구성되어 있다. Object model은 객체를 정의하고 객체 간의 관계를 규명하는데 초점을 두고 있다. 클래스나 객체를 표현하기 위해 object diagram을 사용한다.

Dynamic model은 객체간의 동적인 관계를 표현하기 위해 사용하며 state diagram으로 표현된다. State diagram은 state, event, attribute, condition, action 등으로 구성되어 있다.

State는 객체의 행위에 영향을 주는 객체의 속성값(attribute value)과 연결상태를 나타낸 것으로, 때때로 객체가 만족하는 어떤 조건값이나 지속적인 활동과 관계가 있다. 예를 들면, '0℃에서 100℃사이에 있는 물은 액체상태 이다' 혹은 '전화벨이 울리는 상태' 등을 들 수 있다. Event는 한 시점에서 발생하는 신호로, 한 객체에서 다른 객체로 보내지는 정보의 단방향 전송을 의미한다. Events는 관련된 각각의 event를 요약하여 계층적으로 표현할 수 있고, 이러한 event의 발생순서를 표현하기 위해 scenarios를 이용할 수 있다. Condition은 state와 state에 간에 변환이 일어날 것인지에 대해서 Boolean 함수로 표현된다. Action은 event에 대하여 즉시 일어나는 operation으로 어떤 객체로부터 다른 객체로 보내진다. 특히 action은 entry-exit state와 관련하여 entry action과 exit action으로 나누어 설명할 수 있다. 복잡한 state변화 관리에 nested state diagrams을 이용할 수 있는데, 이들 state와 event는 계층적으로 표현될 수 있다. 객체는 그들 자신의 state를 갖고 독립적으로 작용하며, 다른 객체들과 동시에 발생할 수 있는 속성을 지닌다. 따라서 state diagram에서는 이러한 객체의 동시성을 표현하기 위해 state를 aggreation으로 표현하고 있다.

지금까지 살펴 본 각 방법론에서 시스템의 동적인 측면을 표현하기 위해 사용된 개념 및 표기법에 대하여 [표 1]에 비교·정리하였다. Booch의 방법론에서는 CASE Tool 개발을 목적으로 객체지향에 대한 대부분의 개념들을 기호적 표기법으로 나타내기 보다는 실제 프로그램 언어로의 전환을 용이하게 하는 template로 나타내고 있다. 따라서 각각의 객체지향 개념에 대해 다른 네 개의 방법론과 비교하지 않고 동적표현을 위해 사용된 개념 및 표현 기법을 서술하였다.

Behavior Diagram	Martin & Odell	Desfrey	Embley	Rumbaugh
	Event Diagram	Dynamic Model	Object Behavior Model	Dynamic Model
State	State	State	State	State
Operation	Operation	Method	-	Operation
Event	Event	Event	Action	Event
Contraint	Control condition trigger rule	Condition Relation type	Trigger	Condition Action
Time	-	-	Real-time Choice of prior state	-
Synchronous	Synchronous	Synchronous	Multiple prior-states	Concurrency
Other	Clock	Net class	Exception (state,transition) Path-marker	Activity(state) Attribute(event) Event trace
Complexity	Event type Event partition	Domain Scheme	High-level view State net	Nested state diagram
Senario	Senario	Senario Diagram	-	Senario

*Booch : Class, Object, Relationship, Operation, Message, Visibility symbol, Synchronization symbol, Label, Categroy, Template

[표 1]

행위를 표현한 각각의 방법론들은 기본적인 event, rule 등에 대해서는 공통적인 형태를 가지고 있으며, schema의 복잡도를 관리하는데 있어서는 조금씩 다른 표현방법을 사용하고 있다.

3. 방법론의 제시

객체지향 방법론에서 행위에 대한 표현은 객체를 구현하기 위한 과정을 보다 용이하게 하며, 개념적 모델과 구현에 있어서의 아교적 역할을 수행한다. 또한 소프트웨어공학 측면에서 business model를 분석하거나 표현하기 위해 객체의 구조와 객체의 중요한 변화를 함께 기술해야 한다[Ivar Jacobson 1995]. 객체의 중요한 변화를 의미하는 것은 지금까지 논한 방법들에서 event라고 정의 하고 있다. 위의 각 방법론에서는 정적인 측면과 동적인 측면을 분리하여 기술하였다. 본 연구에서는 행위의 표현을 정적인 모델에 통합하는 새로운 방법론과 표현방법을 제시하고자 한다.

3.1 Static모델과 동적속성과의 통합 구조화 방법

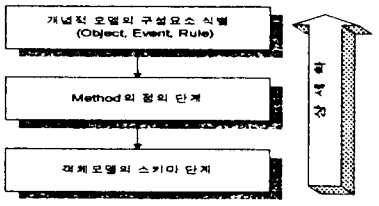
객체의 static 구조에서 동적속성을 함께 표현하기 위해 소프트웨어 공학 측면의 객체지향 기법을 이용한 객체시스템과 실제 응용대상인 business systems간의 관계를 몇 가지로 정리 할 수 있다.

① Business 시스템의 행위는 객체시스템의 행위에 영향을 미친다.[8]

② Business 시스템은 객체시스템을 포함하며, Business 시스템의 모든 요소는 객체시스템의 객체와 event 그리고 rule로 표현된다.[Jacobson 1995]

③ 객체시스템은 business 시스템에 정보를 제공하는데 집중하며, business 시스템은 객체시스템을 통하여 통제와 제어를 실현하며, 이러한 통제와 제어는 의사결정을 지원하는데 사용된다.[8]

객체의 정적 모델에서 동적속성을 포함하기 위해서는 객체와 event를 식별하며, 객체와 이들 events의 발생에 대한 변환 규칙과 처리형태를 정의해야 한다. 이의 각 요소들에 대한 개념적 모델링의 과정을 구조화하기 위해서는 [그림 1]에서 보는 것과 같은 단계가 요구된다.

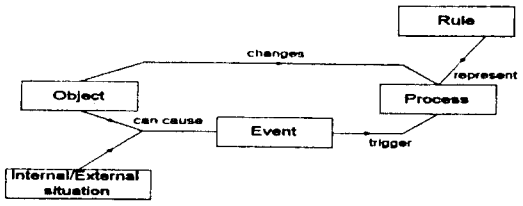


[그림 1]

개념적 동적모델의 구성요소의 식별단계에서는 개념적 동적모델을 구성하는 각각의 요소들에 대한 개념과 이들 요소(object, event, rule)를 정의한다. Method의 정의단계에서는 개념적 동적모델을 구성하는 각 요소들에 대하여 object types, event type과 같은 methods가 정의되며, 이들 method는 서로 다른 객체시스템의 구조적·행위적 측면을 묘사한다. 객체모델의 스키마 단계에서는 객체시스템이 표현되며, 이단계에서 특정 객체시스템에 표현되어야 할 시스템의 행위와 상태에 관련한 이해를 돕는다. 스키마 단계에서는 객체의 상태와 모델내 객체 클래스의 행위를 기술한 후, 개념적인 객체 시스템을 모델화한다. 모델화된 시스템의 구현을 위한 상세화 작업은 앞에서 설명한 각 단계의 역순으로 이루어진다.

전체 시스템의 동적 속성을 표현하기 위해서는 개념적 동적 모델링기법의 요소를 '객체'와 'event' 그리고 이의 변환 '규칙(rule)'과 '프로세스'로 구성할 수 있다. 이의 각 요소들 간의 관계를 간략히 설명하면, 먼저 모든 객체는 어떠한 상태를 가지며 어떤 확실한 상태 혹은 외부 상황의 인식을 통해

events를 발생시킨다. 상태의 변화로 인해 생성된 새로운 객체는 전체시스템과 business시스템의 규칙에 따라 프로세스의 실행을 촉발시킨다. 이러한 프로세스는 객체의 상태를 변화시키는 원인이되며 계속해서 다른 event의 발생원인이 될 수도 있다. 모든 프로세스는 이상적인 상태의 변화를 위해 변환 규칙의 구체적인 예로서 보여진다. [그림 2]에서 이들 각 요소들의 관계를 나타내었다.

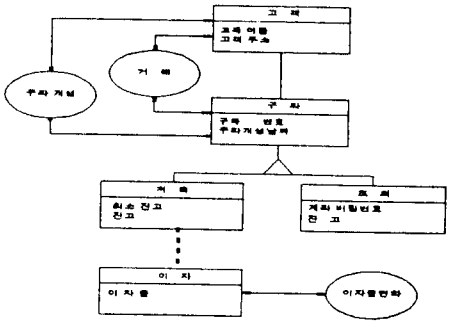


[그림 2]

본 논문에서 제시하는 통합 방법에서 [그림 1]과 [그림 2]를 종합하여 보면, [그림 1]의 방법에 의해 객체, event, rule을 표현하고, 또한 이러한 방법(그림 1)에서 상세화 과정은 [그림 2]에 나타난 process 표현을 용이하게 한다. 이와 같이 본 연구에서는 business 시스템에서 중요한 의미의 변환 즉 event의 발생과 그에 따르는 rule를 가지고 객체의 정적인 모델에 event를 적용하여 실제 응용 표현하고자 하였다.

3.2 Application

본 논문에서 제시한 static모델과 동적속성과의 통합된 방법을 이용하여, [그림 3]에서 실제 은행에서 이루어질 수 있는 업무 일부분을 나타내었다.



[그림 3]

본 모델에서는 한 은행 업무시스템에서 중요한 변화를 의미하는 거래, 구좌개설, 이자율 변동을 등 근 타원 형태인 event로 표현하였다. 이에 따르는 event의 rule이라 할 수 있는 recursive와 singular를 이중 화살표와 기본 화살표로 표현하여, event의 발생이 반복적인지 아니면 일회적인지를 구분하였다. Event에 따라 객체간의 관계가 변화될 수 있는데, 그들의 관계를 static과 temporal한 관계로 나누어 실선과 점선으로 표현하였다. 이러한 구분은 실제 temporal databases의 논리적 설계를 위해 개념적 모델링 단계에서 고려해야 할 것 들이다[3].

4. 결론 및 한계

본 연구는 소프트웨어 공학 측면에서 고려되는 static model 내에 동적요소인 event를 표현함으로써 정적모델과 동적모델과의 자연스러운 연계를 가질 수 있도록 하였으며, 객체, event, rule를 함께 고려하는 방법과 그 표현에 초점을 맞추었다. 또한 business 시스템을 파악하고 분석하는 방법에서 시스템의 구조 즉 static모델과 동적 속성을 함께 표현하는데 관심을 두었다. 이러한 통합 방법에서 static model에 event를 표현한 것이 보다 복잡한 static model를 가지는 것이 아니라 [그림 1]에서 보는 바와 같이 스키마 단계에서부터 구현에 이르기까지 상세화 해 나감으로써 그 복잡성을 해결할 수 있다. 이러한 방법론은 분석, 설계, 구현에 참여하는 최종사용자와 개발자에게 정적모델과 동적모델 간의 일관성있는 표현을 보여주는데 도움을 줄 수 있을 것으로 기대된다. 또한 최근의 객체지향 프로그램언어(C++,Smalltalk 등)가 event 중심의 프로그래밍이기 때문에 기존의 객체지향 방법론에서 제시되어 온 정적모델을 또 다시 event 중심의 프로그램언어로 구현하는데 부딪치게 되는 문제에 도움을 줄 수 있을 것으로 기대된다.

향후 본 연구에서 제시하는 방법론과 그 표현법을 이용하여 완전한 business systems을 표현하고, 그 타당성에 대한 검토를 위한 범례를 완성해야 할 것이다. 이를위해 이들 표기법을 이용한 저작도구의 개발이 우선되어야 하며, 소프트웨어 개발 전 주기를 지원할 수 있는 통합모델로의 확장이 이루어져야 할 것이다. 또한 규모가 큰 시스템의 경우를 지원하기 위한 복잡도 관리에 있어 좀 더 많은 연구가 진행되어야 할 것이다.

참 고 도 서

[1] D. Champeruz & P. Faure, A Comparative Study of Object-Oriented Analysis Methods, Journal of Object-Oriented Programming Vol. 5. NO 1. March /April 1992.

[2] D. Embely, B. Kurtz & S. Woodfield, Object-Oriented Systems Analysis : A Model-Driven Approach, Yordan Press, 1992

[3] Debabrata Dey, Terence M.Barron, Veda C.Storey, A conceptual model for the logical design of temporal database, Decision Support Systems 15, 1995, pp 305-321.

[4] Derek Coleman, Fiona Hayes, & Stephen Bear. Introducing Objectcharts or How to Use Satecharts in Object-Oriented Design, IEEE TRANSACTIONS ON SOFTWARE ENGENEERING, Vol 18, No 1, January 1992.

[5] G. Booch, Object-oriented Design with Application, Benjamin/Cummings, Redwood City, California, 1991. G. C. Murphy, Experiences applying ooa. In TOOLS USA '91, 1991.

[6] I. Graham, Object-Oriented Methods, Addison-Wesely, 1991

[7] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Object-Oriented Modeling and Design, Prentice-Hall, 1991.

[8] Jay Banerjee, Kim won, 외 2명, Data Model Issues for Object-Oriented Applications, ACM, 1987.

[9] Joanne M. Atlee and John Gannon, Sate-based Model Checking of Event-Driven System Requirements, IEEE TRANSACTIONS ON SOFTWARE ENGENEERING, vol 19. No 1, January 1993.

[10] L.J.B. Essink & W.J.Erhart Object modelling and System Dynamics in the Conceptualization Stages of Information Systems Development, IFIP, 1991.

[11] M. Stonbraker, Ed., Reading in Database Systems. San Mateo, CA:Morgan Kaufmann, 1988.

[12] Philippe Desfray, Object-Engineering, Addsion-Wesley Pub. CO. 1994.

[13] S.B.Zdonik and D.Maier, Eds., Readings in Object-Oriented Database Systems. San Mateo, CA: Morgan Kaufmann, 1989.

[14] Shinichi Honiden, et al, An Application of Artificial Intelligence to Object-Oriented Performance Design for Real-Time Systems, IEEE TRANSACTIONS ON SOFTWARE ENGENEERING, VOL 20 . NO 11, NOV 1994.