UML에서 객체 상호작용에 대한 프로세스 대수 접근

(A Process Algebra Approach for Object Interactions in UML)

최 성 운 † 이 영 환 ††
(Sung Woon Choi) (Young Whan Lee)

요 약 객체지향 방법론에서 정적 및 동적 모델에 관한 구문(Syntax)과 의미론(Semantics)의 형식적 정의는 잘 이루어 졌으나 객체 상호작용의 행위에 대한 형식론은 아직까지 제시되지 않았다. 본 논문에서는 객체 상호작용을 묘사하는 UML의 순서(Sequence) 다이어그램을 토대로 프로세스 대수를 사용하여 객체 상호작용을 정의하고 객체 상호작용의 특성을 정규화 시킨다. 이러한 결과는 M. Snoeck과 G. Dedene[9]가 제시한 종속존재 관계의 개념을 상호작용 관계의 개념으로 대체하여 형식론을 전개할 수 있음을 보여준다.

키워드 UML, OOA/D, 프로세스 대수, UP, 형식 명세

Abstract Abstract Formal definitions of syntax and semantics for the static and dynamic models in Object Oriented methods are already defined. But the behavior of interacting objects is not formalized. In this paper, we defined the common behavior of interacting objects in terms of process algebra using sequence diagram in UML and regularized properties of interacting objects. Based on the results, we can develop a formal specification by using of the object interaction instead of the existence dependency suggested by M. Snoeck and G. Dedene[9].

Key words : UML, OOA/D, Process Algebra, Unified Software Development Process, Formal Specification

1. 서 론

객체지향 분석 설계의 중요한 목적 중의 하나는 실세계를 정교하게 묘사하는 것이다. 이를 위해 대부분의 객체지향 개발 방법론에서는 여러 가지 기술들을 결합하였다. 정적 모델에 대한 기술로서 E-R(Entity-Relation) [1] 모델이 널리 사용되었다. 그러나 이 기술은 동적 모델을 묘사하기에는 충분하지 않았다. 그래서 대부분의 객체지향 방법론에서 동적 모델을 위해서 Petri-nets[2], Finite State Machines[3], Harel State Charts[4]와 같은 기술을 사용하였다. UP(Unified Software Development Process)[6]에서도 UML[5]을 사용하여 상태 다이어그램, 활동 다이어그램, 순서 다이어그램 등과 같은 기술을 토대로 동적인 면을 묘사하고 있다. 그러나 이러한 방법론에서 객체 상호작용 모델에 대한 구문과 의미론의 형식적 정의뿐만 아니라 동적 및 정적 상호모델의 일관성 체크도 거의 이루어지지 않았다[7].

우리는 프로세스 대수를 사용하여 객체 상호작용의 구문과 의미론을 전개한다. 프로세스 대수는 모델링에 있어서 동적인 면에 대한 형식론의 기초를 정의할 수 있게 하며 동시성(Concurrency)과 객체 상호간의 커뮤니케이션을 제공한다[8]. 또한 프로세스 대수에 의한 전개는 모순되는 형식론을 배제시키고 수학적 분석을 가능하게 한다. 이를 통하여 특정한 모델링의 스키마를 정의하면 엄밀하며 효과적으로 형식론 전개를 할 수 있음을 보여준다[9].

본 논문에서 제시된 정적 모델은 UML의 클래스 다이어그램(그림4, 5, 6, 7)을 사용하고 동적 모델은 상호

† 비 회 원 : 명지대학교 컴퓨터공학과 교수
choisw@mju.ac.kr
†† 종신회원 : 대전대학교 정보과학부 전산정보보호학전공 교수
ywlee@dju.ac.kr
논문접수 : 2002년 4월 18일
심사완료 : 2002년 11월 25일

작용을 보여주는 UML의 순서 다이어그램(그림1, 2)을 사용한다. 본 논문에서는 객체 상호작용을 전개하기 위해서 순서 다이어그램을 토대로 프로세스 대수를 사용하여 의미론적으로 객체 상호작용을 정의하고 객체 상호작용의 특성을 정규화 시킨다. 특히 클래스 다이어그램과 순서 다이어그램의 일관성을 위해서 두 객체 타입의 관련 이름은 상호작용 연산자에 의해서 생성되는 객체 타입의 이름과 동일해야함을 제안한다. 이러한 결과는 M. Snoeck과 G. Dedene[9]가 제시한 종속존재 관계의 개념을 상호작용 관계의 개념으로 대체하여 간결하게 형식론을 전개할 수 있음을 보여준다.

연구 방법은 순서 다이어그램을 기초로 하여 기존의 프로세스 대수에 의한 정규표현(Regular expression)과 객체의 정의[7, 8, 9]를 토대로 새롭게 객체 상호작용의 의미를 정의한다. 이 정의를 바탕으로 객체 상호작용의 성질을 밝히고 M. Snoeck과 G. Dedene[9]의 결과를 비교 평가한다.

본 논문의 구성은 다음과 같다. 1절에서는 서론으로서 논문의 배경과 연구방법을 소개하고 2절은 관련 연구로서 각각의 방법론에서 사용되는 모델과 이 모델들의 형식론을 언급하며 이어서 프로세스 대수에 의한 접근을 시도한 연구를 소개하고 본 논문의 연구 방향을 제시한다. 3절에서는 순서다이어그램을 소개하고 이것을 본 논문에서 사용하게된 근거를 제시한다. 4절은 객체의 정의와 객체의 행위에 대한 정규표현 방법을 제시한다. 5절에서는 객체 상호작용과 성질을 밝히고 6절에서는 종속존재 사이의 관계와 관련과의 일관성을 제시하며 7절에서 결론을 도출한다.

2. 관련연구

2.1 다양한 방법론에서 사용된 정적, 동적 그리고 객체 상호작용 모델

표 1 방법론에서 사용된 모델 비교 (L:사용빈도 높음, H: 사용빈도 낮음)

Methods, Process/ Notation	저자	정적	동적	객체 상호작용
OSA(L)	Embley et al.	EER. Extension	FSM	Interaction D agram
O/B(L)	Kappel, Schrefl	Own Formalism	Petri-Net	Complex Activity
FUSION (L)	Hayes, Coleman	EER. Extension	STD,Regular Expression	Object Interaction Graph
OMT(L)	Rumbaugh et al.	EER. Extension	STD	Event Trace
OOSA(H)	Schaer, Mellor	EER. Extension	FSM	Object Communication, Access
OOA(L)	Coad, Yordon	EER,G/S,W/P	FSM	Message Connection
OOD(H)	Booch	Variation of EER	STD	Timing Diagram
Catalysis (H)	D'souza, Willis	Unified/OCL	STD/OCL	Sequence Diagram
UP/UML(H)	OMG	Unified/OCL	STD/OCL	Sequence Diagram

표1에서 보듯이 많은 객체지향 소프트웨어 개발 방법론[1, 2, 3, 4, 6, 10, 11, 12, 13, 14]에 있어서 모두 정적모델, 동적 모델 그리고 객체 상호작용의 모델을 사용하고 있다. 주로 동적 모델은 E-R(Entity-Relation)모델을 확장한 EER(Extended Entity Relation)을 사용하였으며 동적 모델은 FSM(Finite State Machine)과 이것을 확장한 STD(State Transition Diagram)를 사용하였다. 그러나 객체 상호작용 모델은 다양하다.

2.2 각 모델에 대한 형식론

E-R 모델과 FSM 형식론은 잘 알려져 있으며 이것을 확장한 EER과 STD도 똑같은 형식론으로 전개되나 객체 상호작용 모델들에 대한 구문과 의미론에 대한 형식론은 이루어지지 않았다[7]. 특히 FSM으로 객체 상호작용의 형식론을 전개하지 못하는 이유는 상태, 전이, 그리고 이벤트들을 첨가하고 제거하거나 재정의 하여 FSM을 세분화시킬 수 없었기 때문이며 FSM을 사용하여서는 행위를 제한하거나 확장 할 수 없어서 메시지 전달에 대한 형식론을 전개할 수 없기 때문이다[9].

UML과 Catalysis에서는 OCL(Object Constraint Language)을 사용하여 제약(Constraint) 조건을 표시하였는데 이것은 자연언어와 형식언어의 중간 형태로서 일반인은 수학적으로 전개되는 형식론을 이해하기 어렵기 때문에 도입된 것으로서 모델의 부가적인 표현이지 형식론을 전개하기에는 부적절하다.

G. Dedene와 M. Snoeck[9]은 종속존재(Existence Dependency)의 개념을 정의하여 객체 타입의 정적 및 동적 의미론 통합에 대한 근거를 마련하였다. 그러나 UML에서는 종속존재라는 관계가 없으며 종속존재를 추가하여 다이어그램을 그리는 방법도 복잡해져서 UML에도 맞지 않는다. 두 객체가 상호작용 할 때 구성되는 상호작용 관계는 우리가 앞으로 전개할 개념으로 보면 종속존재 관계가 된다. 특히 상호작용 관계는 두 객체간의 커뮤니케이션 관련(Communication Association)으로서 대체된다.

2.3 프로세스 대수를 사용한 객체 상호작용에 대한 형식론

G. Dedene와 M. Snoeck[8]은 프로세스 대수를 사용하여 객체 상호작용에 대한 형식론을 전개하였는데 이들은 상호작용 하는 객체 전체의 행위에 관심을 가지고 전개하여 객체지향 개념 스키마에서 데드락(Deadlock) 행위를 제거하는데 적용하였다. 따라서 객체 상호작용에 대한 정의도 이루어지지 않았으며 이 부분의 행위도 파악할 수 없었다. 이들은 종속존재가 되는 새로운 객체를 설정하여서 다이어그램이 복잡해졌으며 불필요한 OET

(Object Event Table)[9]도 사용하였다.

우리는 객체 상호작용을 정의하고 상호작용 하는 부분의 행위를 프로세스 대수로 전개하며 객체 상호작용의 개념과 성질을 밝힌다. 이 방법은 객체 모델을 그대로 유지시키고 상호작용을 나타내는 객체의 이름이 UML의 커뮤니케이션 관련 이름(Association Name)과 동일한 개념임을 제공한다.

3. 이벤트와 순서(Sequence) 다이어그램

표2에서 보듯이 UML에서 객체 상호작용을 나타내는 다이어그램은 순서 다이어그램과 협력(Collaboration) 다이어그램이다. 이 두 다이어그램은 동일한 것을 다른 각도에서 보는 것으로서 같은 의미가 된다[6]. 순서 다이어그램은 시나리오 기반으로서 널리 사용된다. 그러나 형식론이 잘 이루어진 상태차트(Statechart) 다이어그램을 사용한다면 객체 상호작용을 나타내기도 어렵고 상호작용의 의미론도 전개할 수 없다[9]. 따라서 본 논문에서는 순서 다이어그램을 사용하여 객체 상호작용에 대하여 전개한다.

표 2 UML의 뷰(View)와 다이어그램

Structural	Static View	Class Diagram	Class, Association, Generalization, Dependency, Realization, Interface
	Use Case View	Use Case Diagram	Use Case, Actor, Association, extend, Include, Use Case Generalization
	Implementation View	Component Diagram	Component, Interface, Dependency, Realization
	Deployment View	Deployment Diagram	Node, Component, Dependency, Location
Dynamic	State Machine View	Statechart Diagram	State, Event, Transition, Action
	Activity View	Activity Diagram	State, Activity, Completion Transition, Fork, Join
	Interaction View	Sequence Diagram	Interaction, Object, Message, Activation
		Collaboration Diagram	Collaboration, Interaction, Collaboration, Role, Message
Model Management	Model Management View	Class Diagram	Package, Subsystem, Model
Extensibility	All	All	Constraint, Subsystem, Tagged Value

이벤트가 발생하면 메시지가 순차적으로 실행된다. 그림1은 실시간 UML에서 이벤트가 발생하면 메시지로서 객체가 상호작용 하는 구문을 보여준다[15].

우리는 두 객체가 상호작용 하는 부분을 정의하고 상호작용의 행위만 전개하기 때문에 이벤트의 Signature는 고려하지 않으며 이벤트와 메시지를 동일하게 생각한다. 즉, 파라미터를 가지고 있는 메시지도 하나의 이벤트로서 대응시켜서 전개한다. 이 방법은 유용성이 제한될지라도 객체 상호작용의 행위를 표현하는 데에는 간편한 방법이며 프로세스 대수를 사용하여 전개하는데

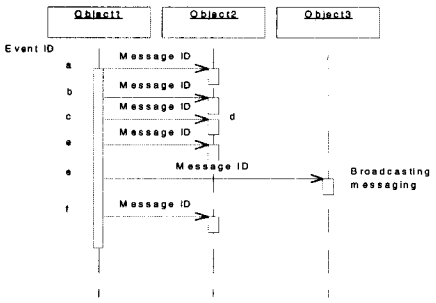


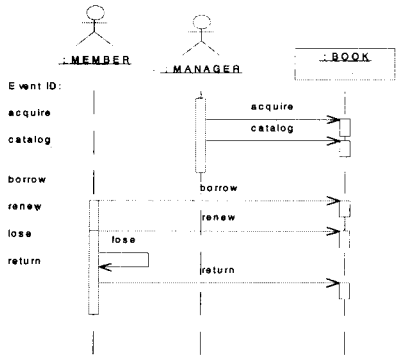
그림 1 순서다이어그램의 구문

잘 적용된다. 특히 객체 상호작용 부분의 행위를 전개하기 위해서 한 이벤트를 발생시키는 객체와 이 이벤트를 받는 객체가 있을 때 이 이벤트는 각각의 객체에 참여한다고 표현한다. 즉 다음과 같이 정의한다.

정의 3.1 (1) 이벤트 e가 객체 P에 참여한다는 의미는 P가 e를 발생시키거나 e를 받는 객체임을 의미한다.

(2) 이벤트 e가 객체 P, Q에 공통으로 참여한다는 의미는 이벤트 e는 P에 참여하고 동시에 Q에도 참여한다는 것을 의미한다. 이때 한 객체는 e를 발생시키고 또 다른 객체는 e를 받는다.

M. Snoeck과 G. Dedene[9]이 제시한 간단한 모델로서 도서관에서 책을 빌리는 예를 들어보자. 그러면 다음과 같이 순서다이어그램이 그려진다.



MEMBER가 발생시키거나 받는 이벤트:
borrow, renew, lose, return
BOOK이 발생시키거나 받는 이벤트:
acquire, catalog, borrow, renew, return
BOOK과 MEMBER에 참여하는 이벤트:
acquire, catalog, borrow, renew, lose return
BOOK과 MEMBER에 공통으로 참여하는 이벤트:
borrow, renew, return

그림 2 간단한 도서관 모델

"MEMBER"는 도서관에서 "BOOK"을 빌린다(borrow). 따라서 이것은 "borrow" 이벤트에 의해서 모델링된다. 이어서 "MEMBER"는 빌린 책을 연장(renew)할 수도 있고 잃어버릴(lose) 수도 있다. 여기서 "MEMBER"와 "BOOK"이 상호작용 하는 부분은 두 객체에 공통된 이벤트로서 "borrow, renew, return"이 참여한다. 우리는 이벤트로서 객체의 행위를 표시하기 위해서 CCS[16], CSP[17], ACP[18]에 의하여 M. Snoeck과 G. Dedene[8, 9]가 전개한 것과 같이 프로세스 대수의 개념을 도입한다.

4. 이벤트에 의한 객체행위의 표현과 객체의 정의

A를 주어진 시스템에 관련된 모든 이벤트들의 집합이라 하자. 그리고 "아무 것도 하지 않는" 이벤트 타입을 "1"로 표시하고 "Deadlock 이벤트 타입"을 "0"로서 표시하자. A위에서 정규표현(Regular Expression)은 "+"(Selection)과 "."(Sequence)에 의하여 만들어진다.

정의 4.1 (1) e가 A 위에서 정규표현 이라는 것은 다음 중의 하나로 표현되는 것을 의미한다:

① $e=0$, ② $e=1$, ③ $e=a$ (여기서 $a\in A$),

④ $e=a+b$ (여기서 $a, b\in A$)

⑤ $e=a.b$ (여기서 $a, b\in A$).

(2) $R(A)$가 A위에서 정규표현들의 집합이라고 하는 것은

$R(A)=\{e|e$는 A위에서 +와 .에 의해서 전개되는 정규표현$\}$으로 정의한다. 여기에서 "+"와 "."는 다음 공리를 만족한다:

$e, e_1, e_2\in R(A)$라 하자. 그러면

① $e+0=e$

② $e+e=e$

③ $e+e_1=e_1+e$

④ $e+(e_1+e_2)=(e+e_1)+e_2$

⑤ $1.e=e=e.1$

⑥ $e.0=0=0.e$

⑦ $e.(e_1.e_2)=(e.e_1).e_2$

⑧ $e.(e_1+e_2)=e.e_1+e.e_2$

⑨ $(e_1+e_2).e=e_1.e+e_2.e$

정의 4.1은 시스템의 행위를 잘 표현해준다. 예를 들어 그림2에서와 같이 도서관에서 서적 예약에 대한 다음의 정의는 성질 (8)에 의하여 동일한 것이다.

RES1 = reserve.cancel + reserve.borrow

RES2 = reserve.(cancel + borrow)

RES1에서는 "reserve" 이벤트가 시스템에 제공되면 "reserve.cancel"와 "reserve.borrow" 중에서 선택이 일어나야 한다. RES2에서는 "reserve" 이벤트에 의해서 예약이 수행되고 이어서 "cancel"과 "borrow"중에서 하나가 수행된다. 즉, 도서관에서 책을 빌리는데 있어서 RES1은 예약하고 취소하거나 예약하고 빌린다는 의미이고 RES2는 예약을 하고 취소하거나 빌린다는 의미이다. 이러한 정규표현은 FSM(Finite State Machine)으로 잘 표현되고 또한 일종의 시나리오와 같아서 정규언어와 똑같이 표현됨을 알 수 있다[9].

⑥번 성질에서 주의할 점을 살펴보자. "deadlock"이 일어나고 나면 그 뒤에는 어떤 행동도 일어날 수가 없다. 따라서 $0.e=0$가 된다. $e.0$은 처음에는 e 처럼 행위를 하고 그런 다음 "deadlock"이 일어나는 프로세스를 표현한다. 이 프로세스는 어떤 의미 있는 행위를 나타낸다고 하여 구현 행위를 전개하는 경우에 있어서의 프로세스 대수의 전개에서는 $e.0\neq 0$으로 전개하기도 한다. 그러나 본 논문에서와 같이 모델링에 의한 객체 상호작용의 경우에서는 단지 정확한 행위만을 취급하기 위해서 $e.0=0$로서 대수적 성질을 만족하는 것으로서 정의한다.

정의 4.2 정의 4.1에서 $R(A)$에 주어진 조건에 덧붙여서 $*$를 정의한다. "$*$"는 반복적인 정규표현을 나타내며 $e\in R(A)$에 대하여 $e^*=\sum_{i\in N} e^i$ 로 정의한다. 여기서 $e^0=1$, $e^1=e$ 그리고 $n\geq 2$에 대하여 $e^n=e.e^{n-1}$을 의미한다. 그리고 $R(A)$의 원소에 대하여 4.1의 공리가 만족되듯이 $R^*(A)$의 원소에 대하여도 4.1의 공리가 만족한다고 가정한다. 이제 $R^*(A)$는 A위에서 $+, ., *$에 의해서 전개되는 정규표현들의 집합이라 정의한다. 우리는 $R^*(A)$를 정규표현 프로세스 대수라 부른다.

정의 4.3 $P(A)$를 A의 부분집합들의 집합 2^A라 하자. $R^*(A)$의 한 정규표현에서 이벤트들을 뽑아내는 함수 $\phi: R^*(A)\rightarrow P(A)$를 다음과 같이 정의한다.

(1) $\phi(1)=\phi$

(2) $\phi(a)=\{a\}$ (여기서 $a\in A$)

(3) $\phi(e+e_1)=\phi(e)\cup\phi(e_1)$ (여기서 $e, e_1\in R*(A)$)

(4) $\phi(e.e_1)=\phi(e)\cup\phi(e_1)$ (여기서 $e, e_1\in R*(A)$)

(5) $\phi(0)=\phi$

우리는 객체의 행위를 표현하기 위해서 객체의 동적인 면만 고려하여 객체를 정의한다. 한 객체는 관련되는 이벤트들과 이들 이벤트에 의한 각각의 정규표현으로

구성된다. 이것을 프로세스 대수를 사용하여 정의하면 다음과 같다.

정의 4.4 한 객체 P는 $\langle \alpha, e\rangle \in \langle P(A), R^*(A)\rangle$로서 정의되며 $e \neq 0$, $\psi(e)=\alpha$을 만족해야 한다. 여기서 $\alpha \subseteq P(A)$는 P와 관련된 이벤트들로서 P의 알파벳이라고 부르며 $e \in R^*(A)$는 P의 행위를 나타내며 P의 정규표현이라고 부른다.

정의 4.5 (1) 한 객체 $\langle \alpha, e\rangle$에서 이벤트들을 뽑아내는 함수를

$C_E : \langle P(A), R^*(A)\rangle \rightarrow P(A)$는 $C_E(\langle \alpha, e\rangle) = \alpha$

로 정의하며,

(2) 한 객체 $\langle \alpha, e\rangle$에서 정규표현을 뽑아내는 함수를

$C_R : \langle P(A), R^*(A)\rangle \rightarrow R^*(A)$는 $C_A(\langle \alpha, e\rangle) = e$

로 정의한다.

예제 4.6 그림2에서 제시한 도서관 모델을 사용하여 "BOOK"과 "MEMBER"에서 각각 이벤트들과 정규표현을 뽑아내면 다음과 같다.

C_E(BOOK)={acquire, catalogue, borrow, renew, return}

C_R(BOOK)=acquire.catalogue.[borrow.[1+renew].return]*

C_E(MEMBER)={borrow, renew, lose, return}

C_R(MEMBER)=[borrow.[1+renew].return]*+borrow.[1+renew].lose

여기서 우리는 "BOOK"의 행위는 구매하고 목록을 정하고 이어서 빌려주고 되돌려 받거나 빌려주고 갱신되고 되돌려 받기가 계속됨을 알 수 있다. 여기에 참여하는 이벤트는 "acquire, catalogue, borrow, renew, return"가 된다. 또한 "MEMBER"의 행위는 빌리고 되돌려 주거나 빌리고 갱신하고 되돌려주기를 반복하거나 또는 빌리고 잃어버리거나 빌리고 갱신하고 잃어버리기를 한다는 것을 알 수 있다. 여기에 참여하는 이벤트는 "borrow, renew, lose, return"가 된다.

객체들은 P, Q, R, ... 등으로 대문자를 사용하거나 $\langle \alpha, e\rangle$와 같이 이벤트의 집합과 정규표현을 쌍으로 표시한다. 선택적 객체의 작용과 순차적 객체의 작용을 정의하기 위해서 "+", ".", "*"를 다음과 같이 정의한다. 두 객체 P와 Q의 선택적 작용은 $P+Q$로 표시하고 순차적 작용은 $P.Q$로 표시한다.

정의 4.7 $\langle \alpha_1, e_1\rangle, \langle \alpha_2, e_2\rangle \in \langle P(A), R^*(A)\rangle$라 하자. 그러면

(1) $\langle \alpha_1, e_1\rangle + \langle \alpha_2, e_2\rangle = \langle \alpha_1 \cup \alpha_2, e_1 + e_2\rangle$

(2) $\langle \alpha_1, e_1\rangle . \langle \alpha_2, e_2\rangle = \langle \alpha_1 \cup \alpha_2, e_1 . e_2\rangle$

(3) $\langle \alpha, e\rangle^i = \langle \alpha, e^i\rangle$

(4) $\langle \alpha, e\rangle^* = \sum_{i \in N} \langle \alpha, e\rangle^i = \langle \alpha, e^*\rangle$

정리 4.8은 정의로부터 쉽게 알 수 있다. 이 성질들은 객체의 선택적 작용과 순차적 작용 시기에 이벤트와 정규표현이 어떻게 표현되는 가를 알려준다.

정리 4.8 P, Q, R이 객체라고 하자. 그러면,

(1) $P+Q = \langle C_E(P) \cup C_E(Q), C_R(P) + C_R(Q)\rangle$

(2) $P.Q = \langle C_E(P) \cup C_E(Q), C_R(P) . C_R(Q)\rangle$

(3) $C_E(P+Q) = C_E(P) \cup C_E(Q)$

(4) $C_R(P+Q) = C_R(P) + C_R(Q)$

(5) $C_E(P.Q) = C_E(P) \cup C_E(Q)$

(6) $C_R(P.Q) = C_R(P) . C_R(Q)$

(7) $(P+Q).R = P.R + Q.R$

5. 객체 상호작용과 성질

M. Snoeck과 G. Dedene[8]이 제시한 M.E.R.O.DE에서와 마찬가지로 UP에서도 객체 상호작용은 순서다이어그램과 관련된 공통 이벤트에 의해서 일어난다. 각 객체의 정규표현에서 상호작용 하는 표현만 뽑아내기 위해서 축소 연산자 " \ "를 정의하자.

정의 5.1 정규표현을 사영(Projection)하는 연산자 "\"는 다음을 의미한다. $B \in P(A)$라 하자.

(1) $0 \setminus B = 0$

(2) $1 \setminus B = 1$

(3) $a \in A$에 대하여 $a \notin B$이면 $a \setminus B = 1$

(4) $a \in A$에 대하여 $a \in B$이면 $a \setminus B = a$

(5) $e_1, e_2 \in R^*(A)$에 대하여

$$(e_1 + e_2) \setminus B = e_1 \setminus B + e_2 \setminus B$$

(6) $(e_1 . e_2) \setminus B = e_1 \setminus B . e_2 \setminus B$

(7) $e^* \setminus B = (e \setminus B)^*$

객체 상호작용은 다음의 규칙을 만족해야 한다.

(1) 객체 상호작용에 참여하는 객체는 UML의 순서다이어그램에 존재하는 객체이어야 한다.

(2) 두 객체가 상호작용 하려면 두 객체에 공통의 참여하는 이벤트가 있어야 한다 (정리 5.2의 (1)과 동일).

(3) P가 Q와 R의 상호작용을 나타내는 객체라고 하면 P에 참여하는 이벤트는 Q와 R각각에 참여해야 되고 P의 정규표현은 Q와 R 중에서 어느 한쪽의 정규표현을 다른 한 쪽에 대하여 사영시

킨 것이어야 한다 (정리 5.2의 (2)와 동일).

이러한 규칙을 만족하도록 객체 상호작용을 정의하면 다음과 같다. 정의 5.2의 (1)과 (2)는 우리가 처음 도입한 개념이고 (3)은 M. Snoeck과 G. Dedene[9]이 제시한 것을 표현을 달리한 것이다.

정의 5.2 (1) 두 객체 $\langle \alpha_1, e_1 \rangle$과 $\langle \alpha_2, e_2 \rangle$가 상호작용 한다는 것은

$$e_1 \setminus \alpha_2 = e_2 \setminus \alpha_1 \neq 0, 1$$

을 만족함을 의미한다.

(2) 두 객체 $\langle \alpha_1, e_1 \rangle$과 $\langle \alpha_2, e_2 \rangle$의 상호작용은 상호작용 연산자 "$\Pi$"를 사용하여 다음과 같이 정의한다.

$$\langle \alpha_1, e_1 \rangle \ \Pi \langle \alpha_2, e_2 \rangle = \langle \alpha_1 \cap \alpha_2, e_1 \setminus \alpha_2 \rangle$$

(3) 두 객체 $\langle \alpha_1, e_1 \rangle$과 $\langle \alpha_2, e_2 \rangle$이 상호작용 할 때 두 객체의 전체행위를 표현하는 객체는 동시성 연산자 "$\|$"를 사용하여 다음과 같이 정의한다.

$$\langle \alpha_1, e_1 \rangle \ \| \langle \alpha_2, e_2 \rangle = \langle \alpha_1 \cup \alpha_2, s \rangle$$

여기서 s는 두 객체 $\langle \alpha_1, e_1 \rangle$과 $\langle \alpha_2, e_2 \rangle$의 전체 행위를 나타내는 정규표현으로서 편의상 $s = e_1 \| e_2$로 표시하고 $a, b \in A$, $e_1, e_2, e_3 \in R^*(A)$라 하면 다음 공리를 만족해야 한다.

① $s \setminus \alpha_1 = e_1$와 $s \setminus \alpha_2 = e_2$를 만족해야 한다.

② $(e_1 + e_2) \| e_3 = (e_1 \| e_3) + (e_2 \| e_3)$

③ $(a.e_1) \| (a.e_2) = a.(e_1 \| e_2)$

④ $(e_1.a) \| (e_2.a) = (e_1 \| e_2).a$

⑤ $a \Pi (b.e_2) = 1$이면 $(a.e_1) \| (b.e_2) = a.(e_1 \| (b.e_2))$

⑥ $(b.e_1) \Pi a = 1$이면 $(b.e_1) \| (a.e_2) = a.((b.e_1) \| e_2)$

⑦ $a \Pi (b.e_2) = 1$이고 $(b.e_1) \Pi a = 1$이면 $(a.e_1) \| (b.e_2) = a.(e_1 \| (b.e_2)) + b.((a.e_1) \| e_2)$

⑧ $(a.e_1) \| 1 = a.e_1$

⑨ $a \| 0 = 0$

⑩ $a \| b = a.b + b.a$

P와 Q가 두 객체일 때 $P \Pi Q$는 두 객체가 상호작용 할 때 공통된 행위를 파악하는 추상객체가 되며 $P \| Q$는 두 객체가 상호작용 할 때 전체의 행위를 파악하는 추상객체가 된다. 다음의 정리는 정의로부터 간단히 증명된다.

정리 5.3 각각의 객체 $P, Q, R \in \langle P(A), R^*(A) \rangle$에 대하여 다음이 성립한다.

(1) $P \Pi P = P$, $P \| P = P$

(2) $P \Pi Q = Q \Pi P$, $P \| Q = Q \| P$

(3) Q는 P의 일부분($Q \subset P$)인 객체라고 하면

$$P \Pi Q = Q \Pi P = Q, \quad P \| Q = Q \| P = P$$

(4) $P \Pi (Q \Pi R) = (P \Pi Q) \Pi R$, $P \| (Q \| R) = (P \| Q) \| R$

(5) $\langle \phi, 1 \rangle \Pi P = \langle \phi, 1 \rangle$, $\langle \phi, 1 \rangle \| P = P$

예제 5.4 예제 4.6에서 살펴보았던 도서관 모델에서 "BOOK"과 "MEMBER"의 상호작용을 알아보자.

C_E(BOOK Π MEMBER)={borrow, renew, return}.

C_R(BOOK Π MEMBER)=[borrow.[1+renew].return]*

C_E(BOOK $\|$ MEMBER)={acquire, catalogue, borrow, lose, renew, return}.

C_R(BOOK $\|$ MEMBER)=acquire.catalogue.[borrow.[1+renew].return]*+borrow.[1+renew].lose

여기서 "BOOK"과 "MEMBER"의 상호작용은 "대여"를 의미하므로 상호작용객체를 LOAN이라 놓자. 즉, LOAN =BOOK Π MEMBER 가 된다. "LOAN"에 참여하는 이벤트는 "borrow, renew, return"이며 행위는 "[borrow.[1+renew].return]*"이 된다. 왜냐하면 acquire. catalogue \ MEMBER = 1이므로,

acquire.catalogue.[borrow.[1+renew].return]*
\MEMBER
= [borrow.[1+renew].return]*

가 성립하기 때문이다.

객체 상호작용은 이벤트들의 연산에 좌우된다. 객체 상호작용과 이벤트의 관계를 알려주는 성질을 생각해 보자. 정리5.5는 두 객체가 상호작용 할 때 선택적 정규표현은 선택적 객체의 작용으로 표현됨을 보여준다.

정리 5.5 $\alpha, \beta \in P(A)$ 이고 $e_1, e_2, e_3 \in R^*(A)$라 하자. 그러면,

$$\langle \alpha, e_1 + e_2 \rangle \ \Pi \langle \beta, e_3 \rangle = (\langle \alpha, e_1 \rangle \Pi \langle \beta, e_3 \rangle) + (\langle \alpha, e_2 \rangle \Pi \langle \beta, e_3 \rangle)$$

증명 : 좌변과 우변의 이벤트들이 같고 정규표현도 같음을 보이면 충분하다.

(1) $C_E(\langle \alpha, e_1 + e_2 \rangle \Pi \langle \beta, e_3 \rangle) = \alpha \cap \beta$

(2) $C_E(\langle \alpha, e_1 \rangle \Pi \langle \beta, e_3 \rangle + \langle \alpha, e_1 \rangle \Pi \langle \beta, e_3 \rangle)$
$= C_E(\langle \alpha, e_1 \rangle \Pi \langle \beta, e_3 \rangle) \cup C_A(\langle \alpha, e_1 \rangle \Pi \langle \beta, e_3 \rangle)$
$= (\alpha \cap \beta) \cup (\alpha \cap \beta)$
$= \alpha \cap \beta$

(3) $C_R(\langle \alpha, e_1 + e_2 \rangle \Pi \langle \beta, e_3 \rangle)$
$= (e_1 + e_2) \setminus \beta = e_3 \setminus \alpha$
$= e_1 \setminus \beta + e_2 \setminus \beta$

(4) $C_R(\langle \alpha, e_1 \rangle \Pi \langle \beta, e_3 \rangle + \langle \alpha, e_1 \rangle \Pi \langle \beta, e_3 \rangle)$
$= C_R(\langle \alpha, e_1 \rangle \Pi \langle \beta, e_3 \rangle) +$

$C_R(\langle \alpha, e_2 \rangle \Pi \langle \beta, e_3 \rangle)$

$= e_1 \setminus \beta + e_2 \setminus \beta = e_3 \setminus \alpha + e_2 \setminus \beta$

정리5.6은 두 객체가 상호작용 할 때 순차적 정규표현은 순차적 객체의 작용으로 표현됨을 보여준다.

정리 5.6 $\alpha, \beta \in P(A)$ 이고 $e_1, e_2, e_3 \in R^*(A)$라 하자. 그러면

$\langle \alpha, e_1 . e_2 \rangle \Pi \langle \beta, e_3 \rangle = (\langle \alpha, e_1 \rangle \Pi \langle \beta, e_3 \rangle) .$
$(\langle \alpha, e_1 \rangle \Pi \langle \beta, e_3 \rangle)$

증명 : 좌변과 우변의 이벤트들이 같고 정규표현도 같음을 보이면 충분하다.

(1) $C_E(\langle \alpha, e_1 \rangle \Pi \langle \beta, e_3 \rangle . \langle \alpha, e_1 \rangle \Pi \langle \beta, e_3 \rangle)$

$= \alpha \cap \beta$

$= C_E(\langle \alpha, e_1 . e_2 \rangle \Pi \langle \beta, e_3 \rangle$

(2) $C_R(\langle \alpha, e_1 . e_2 \rangle \Pi \langle \beta, e_3 \rangle)$

$= (e_1 . e_2) \setminus \beta = e_1 \setminus \beta \cdot e_2 \setminus \beta$

$= C_R(\langle \alpha, e_1 \rangle \Pi \langle \beta, e_3 \rangle . \langle \alpha, e_1 \rangle \Pi \langle \beta, e_3 \rangle)$

객체 상호작용에서 일어나는 이벤트들과 정규표현 방법을 밝혀보자. 정리5.7에서 각각의 성질은 객체가 상호작용 하는 행위를 파악하는데 도움을 준다.

정리 5.7 $a, b, c, d \in A$이고 $a \neq b$이며 $e_1, e_2 \in R^*(A)$라 하자. 그러면,

(1) $\langle \alpha, a . e_1 \rangle \Pi \langle \beta, a . e_2 \rangle = \langle \alpha \cap \beta, a \rangle . (\langle \alpha, e_1 \rangle \Pi \langle \beta, e_2 \rangle)$

(2) $\langle \alpha, a \rangle \Pi \langle \beta, b \rangle = \langle \alpha \cap \beta, 1 \rangle$

(3) $a \notin \beta$이면 $\langle \alpha, a . e_1 \rangle \Pi \langle \beta, d . e_2 \rangle = \langle \alpha, e_1 \rangle \Pi \langle \beta, d . e_2 \rangle$

(4) $a \notin \alpha$이면 $\langle \alpha, c . e_1 \rangle \Pi \langle \beta, a . e_2 \rangle = \langle \alpha, c . e_1 \rangle \Pi \langle \beta, e_2 \rangle$

(5) $c \notin \beta$이고 $d \notin \alpha$이면 $\langle \alpha, c . e_1 \rangle \Pi \langle \beta, d . e_2 \rangle = \langle \alpha, e_1 \rangle \Pi \langle \beta, e_2 \rangle$

(6) $\langle \alpha, c . e_1 \rangle \Pi \langle \beta, 1 \rangle = \langle \alpha \cap \beta, 1 \rangle$

(7) $\langle \alpha, e_1 \rangle \Pi \langle \beta, 0 \rangle = \langle \alpha \cap \beta, 0 \rangle$

증명 : (1) 좌변과 우변의 이벤트들과 정규표현이 같음을 보이자.

$C_E(\langle \alpha, a . e_2 \rangle \Pi \langle \beta, a . e_2 \rangle) = \alpha \cap \beta$

$C_E(\langle \alpha \cap \beta, a \rangle . (\langle \alpha, e_1 \rangle \Pi \langle \beta, e_2 \rangle) = \alpha \cap \beta$

$C_R(\langle \alpha, a . e_2 \rangle \Pi \langle \beta, a . e_2 \rangle) =$
$(a . e_1) \setminus \beta = a . (e_1 \setminus \beta)$

$C_R(\langle \alpha \cap \beta, a \rangle . (\langle \alpha, e_1 \rangle \Pi \langle \beta, e_2 \rangle) = a . (e_1 \setminus \beta)$

(2) $a \neq b$ 이므로 당연하다.

(3) 정리 5.6에 의하여

$\langle \alpha, a . e_1 \rangle \Pi \langle \beta, d . e_2 \rangle = \langle \alpha, a \rangle \Pi \langle \beta, d . e_2 \rangle .$
$\langle \alpha, e_1 \rangle \Pi \langle \beta, d . e_2 \rangle$

이고 $a \notin \beta$ 이므로

$\langle \alpha, a \rangle \Pi \langle \beta, d . e_2 \rangle = \langle \alpha \cap \beta, 1 \rangle = 1$

(4) (3)의 증명과 마찬가지로 증명된다.

(5) (3)과 (4)의 합성으로 증명된다.

(6) $1 \setminus \alpha = 1$ 이므로 당연하다.

(7) $0 \setminus \alpha = 0$ 이므로 당연하다.

정리5.8은 객체의 선택적, 순차적 작용과 상호작용과의 관계를 보여준다.

정리 5.8 $C_E(P) = C_E(Q)$라 하자. 그러면 다음을 만족한다.

(1) $(P + Q) \Pi R = (P \Pi R) + (Q \Pi R)$

(2) $(P . Q) \Pi R = (P \Pi R) . (Q \Pi R)$

(3) $((P + Q) . R) \Pi S = ((P \Pi S) + (Q \Pi S)) . (R \Pi S)$

증명 : $P = \langle \alpha, e_1 \rangle$, $Q = \langle \alpha, e_1 \rangle$라 하자.

(1) $(P + Q) \Pi R = \langle \alpha, e_1 + e_2 \rangle \Pi R$
$= \langle \alpha, e_1 \rangle \Pi R + \langle \alpha, e_2 \rangle \Pi R$
$= P \Pi R + Q \Pi R$

(2) $(P . Q) \Pi R = \langle \alpha, e_1 . e_2 \rangle \Pi R$
$= \langle \alpha, e_1 \rangle \Pi R + \langle \alpha, e_2 \rangle \Pi R$

(3) (1)과 (2)에 의하여

$(P . R + QR) \Pi S = (P . R) \Pi S + (Q . R) \Pi S$

$= ((P \Pi S) + (Q \Pi S)) . (R \Pi S)$

$= (P \Pi S) . (R \Pi S) + (Q \Pi S) . (R \Pi S)$

$= ((P \Pi S) + (Q \Pi S)) . (R \Pi S)$

6. 상호작용, 커뮤니케이션 관련, 그리고 종속존재 사이의 관계

지금까지 우리는 객체의 정의와 객체 상호작용의 성질을 프로세스 대수를 사용하여 전개하였다. 이 절에서는 이것의 응용으로서 객체 타입과 객체 상호작용으로부터 유도되는 추상 객체 타입을 사용하여 관련과의 관계와 종속존재가 되는 객체 타입과의 관계를 연구한다. 우리는 임의의 객체에 관하여 일반적인 행위를 가지고 전개하였음으로 객체 타입 P의 정의도 정의 4.4와 같이 정의되며 두 객체 타입 P와 Q에 대하여서도 객체 상호작용과 마찬가지로 P와 Q의 상호작용 $P \prod Q$가 정의된다.

M. Snoeck과 G. Dedene[9]가 제시한 종속존재(Existence Dependency) 관계로 객체 타입들을 분류하면 로

직이 단순해지고 의미론이 쉽게 전개되지만 객체 타입들은 증가한다. 그러나 객체 상호작용으로 생성되는 객체는 각각의 객체에 종속존재 관계가 됨으로 객체 상호작용 관계로서 종속존재를 대체하면 증가하는 객체 타입을 혼동하지 않게 사용할 수 있고 의미론도 그대로 유지된다. 정의 6.1은 M. Snoeck과 G. Dedene[9]이 제시한 것이다. 정리 6.2는 객체 상호작용에서 생기는 객체를 추상 객체 타입으로 생각하면 종속존재가 됨을 밝힌 것이다.

정의 6.1 P와 Q가 각각 객체타입이라고 하자. P가 Q에 종속존재 관계에 있다고 하는 것은 P의 임의의 한 인스턴스의 생명주기는 Q의 모든 인스턴스의 생명주기에 포함된다는 것을 의미한다. 이때 $P \leftarrow Q$로서 표시한다.

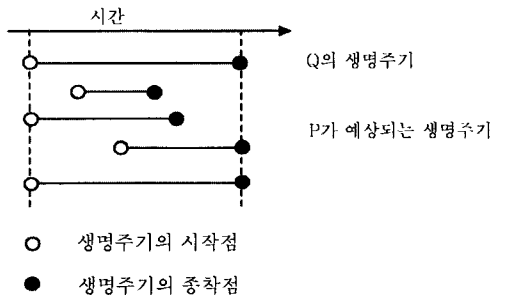


그림 3 종속존재 객체의 생명주기

정리 6.2 P와 Q가 주어진 객체 타입이라면 $P \amalg Q \leftarrow P$ 이고 $P \amalg Q \leftarrow Q$ 이다.

증명 : $C_E(P) \subseteq C_E(Q)$이고 $C_R(Q) \backslash C_E(P) = C_R(P)$이면 P의 알파벳은 Q의 알파벳의 부분집합 관계가 되고 P의 정규표현 또한 Q의 정규표현의 일부분이 되어 P의 행위는 Q의 행위의 일부분이 됨으로 종속존재 관계가 되어 $P \leftarrow Q$가 됨을 이용하여 증명하자. 객체 상호작용의 정의에 의하여 $C_E(P \amalg Q) \subseteq C_E(P)$이고 $C_E(P \amalg Q) \subseteq C_E(Q)$임은 당연하다.

그리고, $C_R(Q) \backslash C_E\ (P \amalg Q) = C_R(P \amalg Q)$와 $C_R(P) \backslash C_E\ (P \amalg Q) = C_R(P \amalg Q)$는 상호작용 연산자의 정의에 의하여 성립한다. 따라서 $P \amalg Q \leftarrow P$ 이고 $P \amalg Q \leftarrow Q$ 이다.

종속존재 관계로 객체 타입들을 분류하면 로직이 단순해지고 의미론이 쉽게 전개되지만 객체 타입들은 증가한다. 그러나 정리 6.2에서와 같이 객체의 상호작용으로 생성되는 객체는 각각의 객체에 종속존재 관계가 됨으로 객체 상호작용 관계로서 종속존재를 대체하면 증가하는 객체 타입을 혼동하지 않게 사용할 수 있고 의미론도 그대로 유지된다.

특히 두 객체가 상호작용 할 때 우리가 전개한 상호작용 연산자 "Π"를 사용하여 생성되는 객체의 이름은 UML에서 관련 이름과 의미론적인 관점에서 동일해야 함을 제시한다. 왜냐하면 관련은 독립적인 객체타입이 아니며 두 객체의 상호작용 관계로서 이름이 부여는 것이기 때문이다[19]. 즉, 우리가 UML을 사용하여 클래스 다이어그램을 그릴 때 모델러의 입장에서 관련의 이름을 명시하지만 이것은 사실 클래스의 실체인 객체들의 상호작용을 의미한다. 우리는 객체 상호작용을 정의하였고 의미론도 전개하였으므로 객체 상호작용의 이름이 관련 이름과 동일해야 한다고 제안한다. 그림4, 5, 6, 7은 참고문헌 [9]에서 종속존재에 관련된 그림을 객체 상호작용 개념을 도입하여 UML 기호를 그대로 사용한 것이다.

(1) 그림4의 모델에서 "EMPLOYEE"와 "DEPARTMENT"의 상호작용 관계로서 ALLOCATION을 설정하자. 그러면
ALLOCATION=EMPLOYEE Π DEPARTMENT
가 되고 ALLOCATION은 두 객체타입의 관련 이름이며 종속존재 관계가 된다.

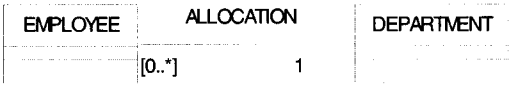


그림 4 고용-부서 모델

(2) 그림5의 모델에서 SERVER와 CLIENT의 상호작용 관계로서 CONNECTION을 설정하자. 그러면
CONNECTION = SERVER Π CLIENT
가 되고 CONNECTION은 두 객체타입의 관련 이름이며 종속존재 관계가 된다.

그림 5 클라이언트-서버 모델

(3) 그림6의 모델에서 PC와 각 객체 타입들이 사용(USE)관계에 있다고 하면 각각의 상호작용은 다음과 같이 각각 관련 이름을 가지며 종속존재가 된다.
PC Π MONITOR = MONITOR USE

PC Π SYSTEM BOX = SYSTEM BOX USE
PC Π KEYBOARD = KEYBOARD USE
PC Π EXTERNAL DEVICE = EXTERNAL DEVICE USE

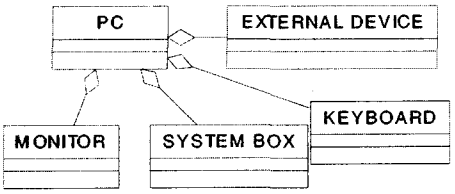


그림 6 PC 모델

(4) 그림7의 모델에서 PEOPLE과 PEOPLE의 상호작용은 PEOPLE이 되어야한다. 그런데 PEOPLE과 PEOPLE이 결혼(MARRIAGE) 관계에 있다면 이 PEOPLE 객체는 MARRIAGE로서만 관련을 한다. 즉, PEOPLE Π PEOPLE = PEOPLE = MARRIAGE

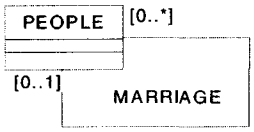


그림 7 PEOPLE 모델

7. 결 론

본 논문은 객체 상호작용 모델에 관한 형식론을 프로세스 대수로서 전개하였다. 우리가 전개한 개념과 성질은 상호작용 하는 부분만의 행위를 파악할 수 있어서 G. Dedene와 M. Snoeck이 프로세스 대수를 사용하여 객체 상호작용의 전체에 대하여 형식론을 전개한 것보다 간결하다. 특히 종속존재의 개념과 OET를 도입하지 않고 UML의 순서 다이어그램을 기초로 하여 전개하였기 때문에 UML의 상호작용 모델에 대한 형식론을 전개한 것이 된다. 본 논문의 주 참고문헌인 Snoneck과 Dedene[7, 8, 9]의 연구 결과와 본 연구의 결과를 비교하면 표3과 같다.

표 3 본 논문의 연구 결과와 Snoeck와 Dedene의 연구 결과 비교

본 논문의 연구 결과	순서 다이어그램	프로세스 대수	객체 상호작용의 정의, 행위 표현, 성질	객체 상호작용의 의미론 전개, 종속존재, 관련과의 연관성 제시	객체상호작용에서 유도된 추상객체 타입= 관련: UML 모델 그대로 유지
Snoeck와 Dedene의 연구 결과	OET /EDG	프로세스 대수	객체 전체의 행위 표현 및 성질	Deadlock 제거, 정적 및 동적 의미론 통합	종속존재 객체 도입: UML과는 다르게 복잡한 모델이 됨

객체 상호작용에 있어서 4, 5절에서 정의와 정리로서 성립하는 일관성 있는 규칙을 요약하면 다음과 같다.

일반적인 규칙 :
- 두 객체의 상호작용에서 생성되는 객체의 이름은 두 객체의 커뮤니케이션 관련 이름과 동일하다.
- 두 객체의 상호작용에서 생성되는 객체는 두 객체에 종속존재 관계가 된다.

이벤트 규칙 :
- 두 객체가 상호작용 하려면 공통으로 참여하는 이벤트가 있어야 한다.
- 두 객체의 상호작용에서 생성되는 객체의 이벤트는 두 객체에 공통으로 참여한다.
- 두 객체에 공통으로 참여하는 이벤트가 있다면 이들 이벤트들은 상호작용에서 생성되는 객체의 이벤트이다.

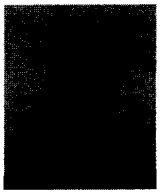
정규표현 규칙 :
- 두 객체의 상호작용에서 생성되는 객체의 정규표현은 두 객체 중에서 어느 한 객체의 정규표현을 다른 한 객체로 사영시킨 것이다.
- 여러 객체가 상호작용 할 때 분배법칙이 성립한다.
- 정규표현의 선택적 순차적 연산("+, ., *")은 객체 상호작용에서도 그대로 유지된다.

우리가 전개한 객체 상호작용에서 생성되는 객체는 UML의 객체사이의 관련과 동일한 개념임을 제안하며 이것은 종속존재 관계가 되고 일반적인 규칙, 이벤트 규칙, 그리고 정규표현 규칙을 만족하므로 모델 사이에서 일관성을 체크하는데 적용될 수 있다.

참 고 문 헌

[1] P. P. Chen, *"The Entity-Relationship Model-Toward a unified view of data,"* ACM Trans. Database Syst. 1(1), 9-16, 1997.
[2] G. Kappel and M. Schrefl, *"Using an object-oriented diagram technique for the design of information systems,"* Dynamic Modeling of Information System, Elsevier Science Publishers, 121-164, 1991.
[3] P. Coad and E. Yourdon, *Object-Orient Analysis,*

Prentice Hall, Englewood Cliffs, New Jersey, 1991.
[4] D. Cloeman, F. Hayes and S. Bear, *"Introducing objectcharts or How to use ststecharts in object oriented design,"* IEEE Trans. Software Eng. 18(1), 9-18, 1992.
[5] I. Jacobson, J. Rumbauch, and G. Booch, The *Unified Software Development Process*, Addison-Wesley, 1999.
[6] UML, *http://www.omg.org and http://www.uml.org.*
[7] G. Dedene and M. Snoeck, *"M.E.R.O.DE.: A Model-driven entity-relationship object-oriented develop ment method,"* ACM Sigsoft, Software Enginee ring Notes, Vol 19, No 3, 51-61, 1994.
[8] G. Dedene and M. Snoeck, *"Formal deadlock elimination in an object oriented conceptional schema,"* Data & Knowledge Engineering Vol 15, 1-31, 1995.
[9] M. Snoeck and G. Dedene, *"Existence Depen dency: The key to semantic integrity between structural and behavioral aspects of object types,"* IEEE Transactions on software Engineering, Vol 24, No 4, 233-251, 1998.
[10] G. Booch, *Object Orient Analysis and Design with Applications*, Second Edition, Benjamin/Cum ming, Redwood City, CA, 1994.
[11] Ivar Jacobson et al. *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1994.
[12] D. W. Embley, B. D. Kurtz, and S. N. Woodfield, *Object-Orient systems analysis : A Model Dri ven Approach*, Yourdon Press, Prentice Hall, Englewood Cliffs, 1992.
[13] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, *Object Oriented Modeling and Design, Prentice Hall International*, Englewood Cliffs, New Jersey, 1991.
[14] S. Shlaer, S. J. Mellor, *Object-Oriented Systems Analysis : Modeling the World in Data*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.
[15] B. P. Douglass, *Real-Tim UML* (Developing Efficient Objects for Embedded Systems) Addison-Wesley , 1998.
[16] R. Milner, *A calculus of communication systems*, Springer, Berlin, Lecture notes in computer science, 1980.
[17] C. A. R. Hoare, *Communicating sequential process*, Prentice-Hall, Series in computer science 1985.
[18] J. C. M. Baeten, *Process algebra*, Kluwer, 1986.
[19] J. Rumbauch, I. Jacobson and G. Booch, *The unified modeling language reference manual*, Addison-Wesley, 1999.



최 성 운
1985년 한국외국어대학교(상학사). 1988년 미국 오레곤주립대학교 컴퓨터공학과(석사). 1992년 미국 오레곤주립대학교 컴퓨터공학과(박사). 1993년~현재 명지대학교 컴퓨터학부 교수, OMG KSIG 회장, 한국정보컨설팅(KIC) 기술자문. 관심분야는 컴포넌트 프레임워크, 객체지향 소프트웨어공학

이 영 환
1982년 충남대학교(학사). 1984년 충남대학교 수학과(석사). 1988년 충남대학교 수학과(박사). 2003년 명지대학교 컴퓨터공학과(박사). 2000년 대전대학교 정보지원센터장. 1988년~현재 대전대학교 정보과학부 전산정보보호학전공 교수. 관심분야는 객체지향 소프트웨어공학, 형식언어, 정보보호