Journal of the Korean Institute of Industrial Engineers
Vol. 27, No. 1, pp. 37-46, March 2001.

객체지향 비즈니스 프로세스 모델링: 계약-협동 네트 모형

김창욱[1†] · 전 진[2] · 김성식[3]

[1]명지대학교 산업시스템공학부/ [2]고려대학교 정보통신기술공동연구소/ [3]고려대학교 산업공학과

Object-Oriented Business Process Modeling: Contract-Collaboration Net Model

Chang-Ouk Kim[1] · Jin Jun[2] · Sung-Shick Kim[3]

[1]Department of Industrial and Systems Engineering, Myong Ji University, Yongin
[2]Research Institute for Information and Communication Technologies, Korea University, Seoul
[3]Department of Industrial Engineering, Korea University, Seoul

Business process(workflow) analysis has been recognized as a core step to building information systems. However, most analysts have recognized that deriving consistent business process artifacts even in a simple business domain is very difficult, since most analysis methods give the designers narrative and/or diagrammatic tools that do not support rigorous analysis rules. This often generates inconsistent analysis results, leading to create useless information systems and resulting in tremendous budget wastes. To overcome this weakness, this paper presents a declarative business process modeling method called contract-collaboration net(cc-net), which is not only somewhat formal but also useful to the practitioners. A case study on bank domain is presented to illustrate our approach.

Keywords : workflow, object-oriented modeling, business process, first order predicate logic

1. 서 론

1.1 연구 배경

현재 많은 국내외의 제조 및 금융 회사들은 그들의 기업 운영관리 정보시스템을 데이터 중심 시스템으로부터 프로세스 중심 시스템(워크플로우 시스템)으로 전환하려는 시도를 하고 있다. 이와 같은 변화는 급변하는 소비자의 요구를 신속하게 반영하여 서비스의 질을 높임으로써 치열한 경쟁이 지속하는 국제 시장에서 생존하기 위한 노력의 일환이다. 데이터 저장 및 검색 기능 자동화 뿐만 아니라 회사 내의 업무 프로세스를 자동화하면 기업 전체의 업무 생산성이 향상되는 것은 자명하다(Taylor, 1995). SAP/R3와 같은 전사적 기업관리 시스템에 프로세스 자동화를 위한 워크플로우 기능이 추가되는 것과 전자 결재 시스템의 등장이 이러한 변화를 단적으로 보여주는 좋은 예이다.

데이터 중심 정보시스템은 다수의 데이터 저장소에 회사 경영에 필요한 데이터를 저장하고 업무를 수행하는 회사원에게 필요한 데이터를 실시간으로 제공하는 역할을 한다. 한편 입출력 데이터 관계에 의해서 업무들의 실행 순서는 정해져 있으며 이를 비즈니스 프로세스라 한다. 따라서 회사 내에는 많은 업무들과 이들의 실행 순서를 통제하는 많은 비즈니스 프로세스가 존재한다. 비즈니스 프로세스의 자동화란 관련 업무들의 실행 순서를 정보 시스템이 통제해 주는 것을 뜻한다. 비즈니스 프로세스의 자동화를 위한 시스템이 프로세스 중심 정보 시스템이며, 이런 시스템은 업무를 자동적으로 연결하고 조정해 줌으로써 회사원이 신속하게 자신이 수행해야할 업무 내용을 파악할 수 있다. 나아가 각 업무를 사람의 간섭 없이 지능적으로 자동화하면 프로세스 자동화의 효과는 극대화될 수

본 연구는 과학기술부 핵심 소프트웨어 사업의 지원을 받아 이루어졌음.
† 연락저자 : 김창욱 교수, 경기도 용인시 남동 산 38-2 명지대학교 산업시스템공학부 Fax : 031-321-6598, e-mail : kimco@mju.ac.kr
2000년 7월 접수, 1회 수정 후, 2000년 11월 게재 확정.

있다.

프로세스 중심 정보시스템을 구축하기 위해서는 두 가지 기술이 적용되어야 한다. 첫째는 프로세스 분석 및 표현 기술이고, 둘째는 구현에 관련된 정보 기술이다. 특히 이중에서 현재까지 취약한 연구 분야가 비즈니스 프로세스 분석 방법에 관련된 것이며, 이는 산업공학 측면에서 보았을 때 시스템 최적화 이전에 수행되어야 하는 시스템 분석 문제로 매우 중요하다. 아무리 최신 인터넷 기술로 프로세스 중심 정보시스템을 구현할지라도 해당 회사의 비즈니스 프로세스를 제대로 파악하고 이를 지원하는 시스템을 구현하지 않았다면 이 시스템은 회사에게 무의미한 것이 되며, 따라서 막대한 예산 낭비를 초래하게 된다.

1.2 연구 목적

본 연구에서는 새로운 객체지향 비즈니스 프로세스 분석 모형인 계약-협동 네트(Contract-Collaboration Net)를 제안한다. 이 모형은 기존의 절차적(Procedural) 프로세스 분석 방법(Ross and Shoman, 1978; DeMarco, 1979)의 단점인 객체 표현의 한계, 프로세스 단위의 불명확성, 사건의 연관성 문제 및 협동 프로세스의 표현의 한계를 극복하고자 제시된 것이며, 다음과 같은 특징을 갖는다.

- 프로세스 수행 주체인 객체를 프로세스 분석 단계부터 명확히 표시하는 모형이다.
- 객체가 협동해야할 대상(객체)을 명확히 표현함으로써 프로세스의 기본 단위를 정의할 수 있다.
- 제약식을 이용하여 사건에 의해서 서로 관련된 프로세스들을 하나의 선언적(Declarative) 메타(Meta) 프로세스로 표현하는 모형이다. 선언적 방법에서는 객체의 상태변수(State Variable or Attribute)들 간에 유지되어야할 일관성(Consistency) 조건을 제약식으로 표현하고 제약식이 파괴되는 사건이 발생할 때 다시 제약식을 만족하는 상태로 복귀하기 위한 과정을 비즈니스 프로세스로 정의한다. 메타 프로세스란 동일 수행 객체 군에서 발생하는 다수의 프로세스를 제약식을 이용하여 하나의 프로세스로 표현하는 것을 의미한다.
- 메타 프로세스로부터 사건-조건-활동 규칙(Event-Condition-Action Rule)을 이용하여 프로세스들을 추출할 수 있기 때문에 능동(Active) 데이터베이스 시스템을 이용하여 바로 구현할 수 있는 모형이다.

본 논문은 다음과 같이 구성되어 있다. 2절에서는 분석 예제인 은행 수신 도메인과 함께 계약-협동 네트 모형을 소개한다. 3절에서는 계약-협동 네트 모형과 비교하여 기존 연구의 문제점을 다룬다. 4절에서는 결론 및 추후 연구 과제를 언급한다.

2. 계약-협동 네트 모형

2.1 은행 수신 업무 도메인

계약-협동 네트 모형을 예제와 더불어 설명하기 위해서 우선 은행 수신 업무 도메인을 설명하기로 한다. 은행 수신 업무는 고객의 입출금에 관련된 업무를 뜻한다. 고객은 은행에서 금융 상품에 하나 이상 가입할 수 있고, 가입 시 계좌를 개설해야 한다. 계좌 개설은 은행원, 대리, 책임자 등의 업무권한이 다른 직원을 거쳐 개설이 승인된다. 또한 계좌의 거래 내역은 반드시 원장에 기록되어 있어야 하며, 무통장 계좌를 제외한 모든 계좌는 통장이 있어야 한다. 한 통장에 여러 계좌가 기재되기도 한다. 한 계좌의 입출금 한도액은 상품에 의하여 결정되며, 고객의 신용상태에 따라서 계좌의 출금 한도액이 조정될 수도 있다.

계좌 간에는 정기적으로 계좌이체를 하기 위한 자동이체가 설정될 수 있으며, 정해진 기간동안 거래가 없는 계좌는 잡좌로 정리되어 고객의 요청이 있기 전까지는 거래를 할 수 없으며 원장도 삭제된다. 고객의 주소가 바뀌거나 거래점을 옮기는 경우 은행 간에는 계좌의 이관과 이수가 발생한다. 이러한 은행 수신 업무에 대한 지식을 바탕으로 비즈니스 프로세스 모델링을 수행하기 위하여 다음 절에서 비즈니스 시스템의 구성요소를 정의한다.

2.2 비즈니스 시스템 구성요소

정보시스템 모델링의 핵심은 현실 비즈니스 시스템의 구성요소를 얼마만큼 자연스럽게 정보 모델에 완벽하고 추상적으로 반영하느냐에 있다. 절차적 방법론과 같이 단지 업무 요소만을 정보 모델에 반영하다보면 프로세스 묘사가 매우 부자연스럽고 난해하게 되기 쉽다. 따라서 본 연구에서는 우선 현실 시스템의 구성요소를 객체지향 관점에서 분석하기로 한다. 객체지향 관점에서 볼 때 현실 비즈니스 시스템은 다음과 같은 요소로 구성되어 있다.

- 객체(Object) 및 클래스(Class) 집합 - 객체는 시스템의 핵심 구성요소로서 업무 집행 역할 및 업무에 필요한 문서 등 데이터를 의미한다. 각 객체는 상태변수 집합과 메쏘드(Method) 집합으로 구성된다. 클래스는 같은 유형의 객체들을 추상화한 것이다. 따라서 각 객체는 하나의 클래스의 인스턴스(Instance)로 볼 수 있다. 예를 들어, 은행 수신 업무 도메인에서 클래스로는 고객, 은행원, 대리, 지점장, 계좌 등이 존재하며, 객체는 특정 고객, 특정 대리를 의미한다.
- 사건(Event) 집합 - 사건은 객체에 의해서 발생한다. 시스템에서 발생하는 사건을 일반화(Generalization)하면 객체 추가, 삭제 및 상태변수 갱신으로 구분할 수 있다. 예를 들어, 신규 고객 유치, 기존 고객 삭제 및 기존 고객 정보 변

경 등은 모두 고객 객체로부터 발생하는 사건이다. 또한 객체에서 발생하는 사건 간에는 선행 관계가 있다. 사건에 의해서 진행되는 프로세스는 여러 개의 객체에 의해서 수행된다. 프로세스 진행 중에 수행 객체에 의해서 발생하는 사건을 종속 사건이라 하고 종속되지 않은 사건을 독립 사건이라고 정의한다. 예를 들어, 신규 고객 사건은 독립 사건이고, 이에 대응되는 계좌 개설 프로세스를 진행 중에 발생할 수 있는 대리의 부재 등은 종속 사건이 된다.

- 업무(Task) 집합 - 객체의 메쏘드에 해당한다.
- 프로세스(Process) 집합 - 비즈니스 프로세스는 사건과 일대일 대응 관계에 있다. 예를 들어, 신규 고객 삽입 사건에 대응되는 것은 계좌 개설 프로세스이다. 각 프로세스는 수행 객체들의 메쏘드 호출 순서에 의해서 결정된다.

비즈니스 시스템 구성요소들의 상관 관계를 설명하면 다음과 같다(<그림 1> 참조). 객체는 사건을 발생시키며 이 사건에 의해서 비즈니스 프로세스가 시작된다. 수행 객체들의 메쏘드 호출 순서에 의해서 비즈니스 프로세스는 진행되는 데 수행 객체에서 종속 사건이 발생하면 새로운 비즈니스 프로세스가 생성된다. 이런 관계로 볼 때 비즈니스 프로세스 모델링은 객체를 중심으로 행해져야함을 알 수 있다. 기존의 절차적 방법론은 시스템 구성요소 중에서 단지 업무를 위주로 프로세스를 묘사하다보니 객체와 종속 사건을 제대로 반영하지 못하였다. 본 연구에서 제시하는 계약 - 협동 네트 모형은 비즈니스 프로세스 분석 초기부터 객체를 중심으로 프로세스를 묘사하며 선언적 방법이므로 독립 사건과 종속 사건의 관계가 하나의 메타 프로세스에서 설명될 수 있다는 장점이 있다.

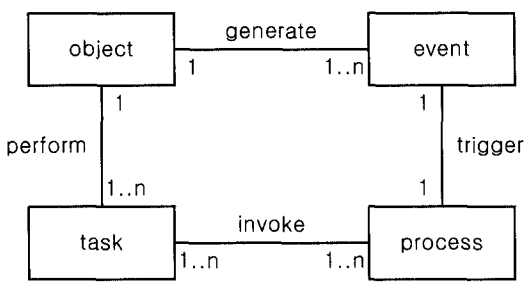


그림 1. 객체, 사건, 메쏘드 및 프로세스 관계.

2.3 계약 - 협동 네트 정의

계약 - 협동 네트 모형은 다음과 같은 두 가지 가정 하에서 출발한다.

[가정 1] 비즈니스 프로세스의 진행은 객체 간의 *계약*과 *협동*과정의 연속이다.

[가정 2] 각 객체는 계약/협동 대상 객체를 제외한 다른 객체와는 *독립*적으로 업무를 수행한다.

[가정 1]은 Petri net(Murata, 1989)이 Place와 Transition으로 절차적 프로세스를 기술하는 양식과 비슷하다. 그러나 Petri net이 업무를 Transition으로, 업무 결과 및 상태를 Place로 표현하여 프로세스를 기술하는 반면, 계약 - 협동 네트는 객체들 간에 계약을 하고 계약 사항을 완수하기 위해서 협동을 하는 과정을 비즈니스 프로세스로 정의한다. 계약 조건 및 협동 조건은 객체들의 상태변수들을 이용하여 제약식으로 표현한다. [가정 2]는 객체의 시계 범위(Visibility Scope) 또는 메쏘드 호출 범위를 뜻하는 것으로 각 객체는 계약과 협동 관계에 있는 객체를 제외한 다른 객체와는 상호 호출을 하지 않는다는 것을 의미한다.

이런 두 가지 가정아래 계약 - 협동 네트 모형을 정형화하여 표현하면 다음과 같다.

$$CC-Net = \langle TS^d, TS^r, TS^c, CRS, CFS, CS \rangle \quad (1)$$

where TS^d : 도메인 클래스 A^d의 집합
TS^r : 참조 클래스 B^r의 집합
TS^c : 계약 클래스 C^c의 집합
CRS : 계약 의존 관계 $\Rightarrow$의 집합
CFS : 계약 함수 $\rightarrow$의 집합
CS : 제약식 집합

계약 - 협동 네트 모형을 설명하기 위해서 다음과 같은 용어를 정의한다. 우선 객체를 영어 소문자로, 클래스를 영문 대문자로 표기한다. 예를 들어, 객체 a는 클래스 A의 인스턴스를 의미한다. 또한 객체 a는 상태변수 집합 $S(a)=\{a.s_1, a.s_2, \cdots, a.s_l\}$을 갖는다.

[정의 1] 도메인 클래스(Domain Class)
도메인 클래스는 스스로 사건을 유발시키는 능동 클래스를 뜻한다(Self-Event Generation). 즉, 자기 스스로 삽입, 삭제, 상태변수 갱신 사건을 생성하는 객체 종류를 도메인 객체라고 정의한다. 만일 클래스 A가 도메인 클래스이면 A^d로 표기한다.

[정의 2] 참조 클래스(Reference Class)
참조 클래스는 스스로 사건을 생성해 낼 수 없는 수동 객체의 일종이며, 객체 간의 계약과정에 참여하여 계약 객체를 생성할 때 필요한 데이터를 제공하는 객체를 뜻한다. 만일 클래스 B가 참조 클래스이면 B^r로 표기한다.

[정의 3] 계약 클래스(Contract Class)
계약 클래스도 수동 객체의 일종이며, 객체 간의 계약 함수로서 생성되는 객체이다. 만일 클래스 C가 계약 클래스이면 C^c로 표기한다.

비즈니스 시스템 내에 존재하는 모든 클래스는 도메인 클래스이거나 계약 클래스, 또는 참조 클래스이어야 한다. 은행 수

신 업무에서 도메인 클래스 집합은 {고객, 은행, 은행원}이며 이들은 스스로 삽입, 삭제, 상태 변수의 변경이 가능하다. 예를 들어, 은행원의 경우 삽입은 신규 채용을 뜻하며, 삭제는 해고, 상태 변경은 휴가, 결근 등을 뜻한다. 반면에 계약 클래스 집합은 {계좌, 통장}이며, 이 객체들은 고객, 은행원의 행위에 종속되어 추가, 삭제, 변경 사건이 발생하는 것이지 독립적으로 사건을 생성하지는 못한다. 또한 은행 수신 업무에서 참조 클래스는 {상품}이며, 고객과 은행원의 계약 생성 시 참조되어 계좌 객체를 생성하는 데에 도움을 주는 객체이다.

[정의 4] 계약 의존 관계(Contract Dependency Relationship)

계약 의존 관계 ⇒는 방향성이 있는 관계로 방향은 계약의 진행 방향을 뜻한다. 즉, $C \Rightarrow A, B$는 계약을 요구하는 클라이언트 클래스(C)로부터 계약을 받아들이는 서버 클래스(A, B)로 계약 방향이 존재한다.

[정의 5] 계약 함수(Contract Function)

계약 함수 →는 계약 의존 관계에 있는 객체들에 의해서 하나의 계약 객체를 생성하는 함수 관계를 뜻한다. 즉, $A, B, C \rightarrow D^c$는 계약 의존 관계에 있는 클래스(A, B, C)로부터 계약 조건을 명시하는 계약 클래스(D^c)를 생성함을 뜻한다. 이때 참조 클래스(E^r)가 존재한다면 $\rightarrow_{(E^r)}$으로 표기하여 참조 클래스를 가진 계약 함수로 표현한다.

예를 들어, 은행 수신 업무에서 계약 의존 관계로는 고객(클라이언트 객체)과 은행원(서버 객체)에서 발생하는 신규 계좌 개설 관계가 있으며, 이때 상품(참조 객체)을 참조하는 계약 함수에 의해서 생성되는 계약 객체는 계좌가 된다.

계약 의존 관계에서 도메인 객체 또는 계약 객체 모두 클라이언트 객체로 참여할 수 있으나, 서버 객체로서 참여하는 데에는 제약이 따른다. 이에 대한 이유는 2.4절 막힘현상에서 설명하겠다.

<그림 2>는 은행 수신 업무에 대한 계약-협동 관계를 다이어그램 형태로 표현한 것이다. 실선 화살표는 계약 의존 관계를, 점선 화살표는 계약 함수를 나타내고 있다. 계약 의존 관계를 표현하는 실선 화살표에서는 화살표의 방향으로 클라이언트 객체와 서버 객체(→ 방향)가 구분되며 계약 당사자 간의 참여 객체 수를 숫자로 표현할 수 있다. 참여 객체 수가 표시되어 있지 않은 경우는 참여 객체 수가 1인 경우이다. 그리고 점선 화살표 위의 원을 한쪽 끝으로 하는 선은 참조 관계를 의미한다. 계약 함수와 의존 관계의 참여 객체 수는 각각 함수적 특징과 참조의 특성상 모두 1로 제한된다. 그림에서 계좌에서 통장으로의 계약 의존 관계에서 무통장 계좌의 경우를 위하여 참여 객체 수를 0.1로 정의하였다.

[정의 6] 일관성 제약식(Consistency Constraint)

제약식은 객체 상태변수들 간에 만족해야할 관계를 일차 술

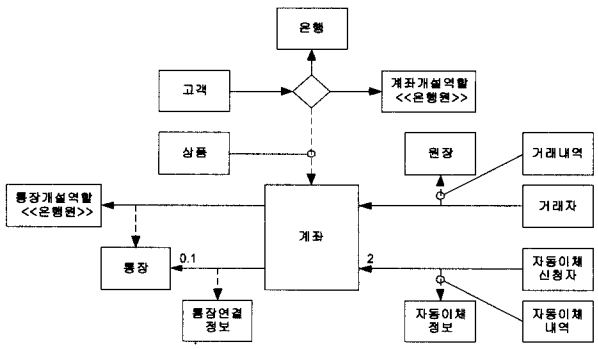


그림 2. 은행 수신 업무의 계약-협동 네트.

어 논리(First Order Predicate Logic)를 이용하여 표현한 것이다. 따라서 비즈니스 시스템이 정상 상태에 있을 때는 모든 객체 상태변수 간의 제약식이 만족한다. 일관성 제약식은 *만족해야할 시점에 따라서* 다음과 같이 세 종류로 구분한다.

(1) C_1 제약식 : 계약 의존 관계를 생성하는 시점에서 만족해야할 제약식을 의미한다. C_1 제약식의 일반적인 형태는 식 (2)와 같다. C_1 제약식은 두 조건식(Implication Condition)의 AND 조합으로 나타난다. 첫번째 조건식은 클라이언트 객체의 계약 가능조건($P_1(a)$)이 만족하면 이러한 계약 요구를 만족하기 위하여, 서버 객체 자체의 계약 가능조건($P_2(b)$)이 만족하는 서버가 $f_1(a)$의 수 만큼 필요함을 나타내고 있다. 이때 설정되는 클라이언트 객체와 서버 객체 간의 계약객체 생성을 위한 관계식으로 $P_3(a, b)$ 를 설정한다. 식 (2)의 두번째 조건식은 첫번째 계약관계 설정에 의한 계약 객체 생성에 관한 제약식이다. 즉, 첫번째 조건식에서 설정된 $P_3(a, b)$ 관계에 있는 모든 클라이언트 객체와 서버 객체들의 조합은 참조 객체 $c:C^r$을 참조하여 계약 객체 $d:D^c$를 생성하고 있음을 표현하고 있으며, 생성된 계약 객체는 관계식 $P_4(a, b, c, d)$를 만족해야 한다.

$$\begin{aligned} &\forall a:A\,\{P_1(a) \rightarrow \exists_{f_1(a)}\, b:B, P_2(b) \ \wedge\ P_3(a,b)\} \wedge \\ &\forall a:A, \forall b:B, \forall c:C\,\{P_3(a,b) \rightarrow \exists d:D, P_4(a,b,c,d)\} \end{aligned} \quad (2)$$

C_1 제약식의 예로서 은행 예제의 고객의 계좌개설 관련 제약식을 살펴보면 다음과 같다.

$$\forall c : \text{고객}, \text{거래신청}(c), \neg(c.\text{신용상태} = \text{'불량'})$$

$$\rightarrow \exists b : \text{은행}, \exists w : \text{은행원}, \text{거래신청 관계}(c, b, w)$$

우선 첫번째 조건식에 해당하는 부분은 고객의 거래신청이 있고, 거래 가능한 신용상태를 가진 경우 해당 은행, 은행원이 존재해야 한다는 조건이다. 그러나 대부분의 경우 고객의 거래신청에 대하여 서버 객체들이 없는 경우는 없다. 그런 경우라면 서비스가 불량한 은행에서의 은행원 부재정도일 것이다.

다음으로 계약 객체인 계좌가 생성(개설)되는 부분의 제약

식을 살펴보면 다음과 같다.

$\forall c$:고객, $\forall b$:은행, $\forall w$:은행원, $\forall g$:상품,
거래신청관계(c, b, w)

$\rightarrow \exists a$:계좌, 계좌개설관계(c, b, w, g, a)

(2) C_2 제약식 : 협동 기간(계약을 유지하는 기간) 동안에 유지되어야할 제약식을 의미하며 식(3)과 같이 표현된다.

$$\forall a{:}A, \forall b{:}B, \forall c{:}C, \forall d{:}D, P_4(a, b, c, d), P_5(a, b, c, d)$$
$$\forall a{:}A, \forall b{:}B, \forall c{:}C, \forall d{:}D, P_4(a, b, c, d), P_5(a, b, c, d)$$
$$\leftrightarrow P_5(a, b, c, d)@after(t) \quad (3)$$

where $P_5(a, b, c, d)$: 협동관계 불변조건(Collaboration Invariant)

$P_5(a, b, c, d)@after(t)$: t 시간이 흐른 뒤의 협동관계 불변조건

주로 시간의 변화에 따른 객체 상태 변경(예를 들어, 이자계산)시 적용

예를 들어, 모든 계좌는 출금 한도액이 존재한다. 출금, 이체 등으로 인하여 계좌의 상태(잔고)가 변경되어도 해당 계좌의 출금 한도액 미만으로 내려갈 수 없다. 이러한 계좌의 출금 한도액은 참조 객체인 상품에 의해서 정의된다.

$\forall g$:상품, $\forall a$:계좌, 계좌개설 관계$(*, *, *, g, a)$,
$(a.$잔고$\geq g.$출금한도액$)$

또는 이자 계산 시점이 되어 이자가 계좌의 잔고에 누적되어야 한다. 이러한 시간에 따른 협동단계의 불변조건은 객체 a의 t시점 이전 상태($a@before(t)$)와 t시점 이후 상태($a@after(t)$)를 이용하여 제약식을 구성한다. t가 표기되지 않은 경우에는 제약식 적용 바로 전상태와 후상태를 의미한다. 이자 계산도 상품의 특성에 의해서 정의되며 계좌의 상태가 잡좌가 아닌 경우에만 적용된다.

$\forall g$:상품, $\forall a$:계좌, 계좌개설 관계$(*, *, *, g, a)$,
$\neg(a.$상태$=$'잡좌'$)$, $(Tnow = g.$이자계산일$)$

$\leftrightarrow (a.$잔고$= a.$잔고$@before + g.$이자계산$(a.$잔고$@before))$

where $Tnow$: 현재시간

(3) C_3 제약식 : 계약을 종료할 때 만족해야할 제약식을 의미하며 식(4)와 같이 표현된다.

$$\forall a{:}A, \forall b{:}B, \forall c{:}C, \forall d{:}D, P_4(a, b, c, d) \rightarrow P_6(d) \quad (4)$$

where $P_6(d)$: Collaboration Termination Condition

예를 들어, 계좌 이체는 신청할 때 명시한 계약 만료일(명시하지 않으면 신청일로부터 1년)이 지나면 효력이 없어지므로 계좌 이체라는 계약이 완료되며 계좌 이체 정보라는 계약 객체도 소멸된다.

$\forall ct$:자동이체신청자, $\forall a$:계좌, $\forall ts$:자동이체내역,
$\forall at$:자동이체정보, 자동이체관계(ct, a, ts, at)

$\rightarrow (at.$계약만료일$= Tnow)$

[정의 7] 협동 프로세스(Collaboration Process)

협동 프로세스는 계약-협동 네트의 최소 단위로서 계약의존 관계 및 계약함수에 있는 객체들의 집합과 제약식으로 정의한다. 클래스 A 에서 클래스 B로의 계약의존 관계가 있고 이들 클래스들과 참조 클래스 D^r에 의해서 계약 클래스 C^c가 생성되는 협동 프로세스 P는 $\{\{A \Rightarrow B\} \rightarrow_{(D^r)} C^c\}$로 표기한다.

협동 프로세스의 개념은 데이터 베이스 함수 종속 관계(Functional Dependency)에서 제시된 Maximal Object 개념(Maier and Ullman, 1983)과 유사하다. <그림 2>의 계약-협동 네트에서 존재하는 협동 프로세스는 다음과 같다.

$P_1 = \{\{$고객$\Rightarrow$은행, 은행원$\} \rightarrow_{(상품)}$계좌$\}$

$P_2 = \{\{$계좌$\Rightarrow$은행원$\} \rightarrow$통장$\}$

$P_3 = \{\{$계좌$\Rightarrow$통장$\} \rightarrow$통장연결정보$\}$

$P_4 = \{\{$거래자$\Rightarrow$계좌$\} \rightarrow_{(거래내역)}$원장$\}$

$P_5 = \{\{$자동이체신청자$\Rightarrow$계좌$\} \rightarrow_{(자동이체내역)}$자동이체정보$\}$

하나의 협동 프로세스는 메타 수준의 프로세스를 표현한 것이기 때문에, 동적으로 변화하는 프로세스의 진행 과정(실제 프로세스)을 설명할 필요가 있다. 이를 위해서 먼저 다음과 같이 사건의 종류를 구분한다.

[정의 8] 사건(Event)

사건은 제약식을 파괴하는 객체 a의 상태변수 전이(Transition)를 의미한다. 다시 말해서 상태변수 전이 쌍 $\langle S(a), S'(a)\rangle$를 사건으로 보며, 여기서 상태 $S'(a)$는 시스템내의 제약식을 하나 이상 파괴하는 상태이다. 상태변수의 전이를 사건으로 보았을 때 발생할 수 있는 사건 종류는 객체 삽입, 삭제, 상태변수 변경이 있다. 한편 도메인 객체 중에서 비즈니스 시스템 내부에 있는 객체에 의해서 발생하는 사건을 내부(Internal) 사건, 시스템 외부에 있는 객체에 의해서 발생하는 사건을 외부(External) 사건으로 정의했을 때, 2.2절에서 언급한 독립 사건은 외부 사건 중에서 삽입 사건으로 정의하며, 그 밖의 사건은 모두 종속 사건으로 정의한다. <표 1>은 사건 분류 기준을 보여준다.

표 1. 사건의 분류

사건 분류	외부 사건	내부 사건
생성	독립 사건	종속 사건
삭제	종속 사건	종속 사건
갱신	종속 사건	종속 사건

예를 들어, 고객은 은행 시스템 외부에 있는 객체이므로 고객 객체로부터 발생하는 고객 추가 사건, 기존 고객의 계약 해지(삭제) 및 상태변수 갱신 등은 외부 사건이다. 이 중에서 고객 추가 사건은 독립 사건이기도 하다. 독립 사건은 모든 프로세스의 시발점이다. 다시 말해서 외부 객체가 시스템 내부 객체와 처음으로 계약을 맺으려면 그 외부 객체가 삽입 사건을 발생해야만 하며, 나머지 사건 종류인 내부 사건과 외부 객체의 삭제, 갱신 사건은 독립 사건에 의해서 파생되는 종속 사건이다.

협동 프로세스의 진행 과정인 협동 프로세스의 생명 주기(Life Cycle)는 다음과 같은 절차를 따른다.

Step 1. 계약 단계 : 독립 사건이 발생하면 사건을 일으킨 클라이언트 객체와 한 개 이상의 서버 객체는 계약 의존 관계를 맺어야하며 이때 계약 시점 제약식 (C_1제약식)을 만족해야만 계약이 성사된다. 이 경우

(1) C_1 제약식이 만족하면 계약이 성사된다.

(2) C_1 제약식이 만족하지 않는다면 서버 객체가 종속 사건을 발생하여 제약식이 만족할 때까지 기다린 후 계약을 성사시킨다. 또는

(3) 클라이언트 객체가 독립 사건을 취소하는 사건을 발생하여 C_1 제약식을 만족시킨다.

Step 2. 협동 단계 : 계약이 성사되면 계약 사항을 명시하는 계약 객체가 계약함수에 의해서 생성되며, 클라이언트 및 서버 객체들은 협동 단계에 들어선다.

Step 2-1. 협동 단계 중에 C_2 제약식이 파괴되는 종속 사건이 발생하면

(1) C_2 제약식이 만족되도록 계약의존 관계에 있는 객체들 간에 상태변수를 조정(갱신 종속 사건)하거나, 또는

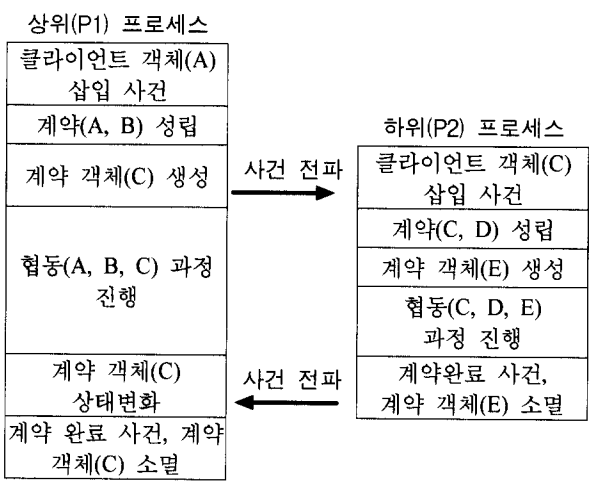
(2) Step 1로 돌아가서 다시 계약을 하거나 계약 취소를 한다.

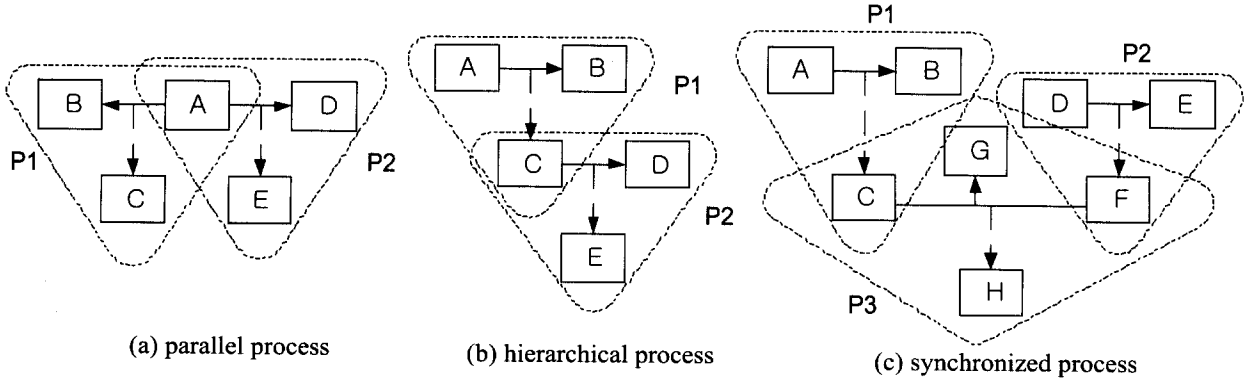
Step 2-2. 협동 단계 중에 C_3 제약식이 만족되는 종속 사건이 발생하면 계약과 협동은 성공적으로 끝난다.

계약-협동 네트 모형에서는 최소 프로세스 단위인 협동 프로세스가 다른 협동 프로세스와 연결될 수 있는 데 연결 형태에 따라서 <그림 3>과 같이 병렬(Parallel) 프로세스, 계층적(Hierarchical) 프로세스 및 동기화(Synchronized) 프로세스로 구분된다.

(1) 병렬 프로세스 : 계약-협동 네트가 두 개의 협동 프로세스 $P_1=\{\{A\Rightarrow B\}\rightarrow C\}$과 $P_2=\{\{A\Rightarrow D\}\rightarrow E\}$로 형성된 경우 A의 독립 사건에 의해서 P_1에서 B와 P_2에서 D와 동시에 계약을 맺는 관계가 되기 때문에 이 프로세스들은 병렬로 진행될 수 있다(<그림 3(a)> 참조). 즉, 하나의 클라이언트 객체로부터 사건이 발생될 때 그 사건이 두 개 이상의 계약을 동시에 진행시킬 때 발생하는 경우이다.

(2) 계층적 프로세스 : 계약-협동 네트가 두 개의 협동 프로세스 $P_1=\{\{A\Rightarrow B\}\rightarrow C\}$과 $P_2=\{\{C\Rightarrow D\}\rightarrow E\}$로 형성된 경우 P_1의 계약 클래스 C의 생성(삽입) 사건에 의해서 P_2의 D와 계약의존 관계가 진행되는 경우를 뜻한다(<그림 3(b)> 참조). 즉, 상위 협동 프로세스가 협동 과정 중에 다시 계약 객체가 다른 도메인 객체와 하위 협동 프로세스를 형성하는 경우이다. 계층적 프로세스는 병렬 프로세스와 달리 두 개의 프로세스가 독립적으로 진행하지 않으므로 상위 및 하위 프로세스의 생명 주기는 다음과 같은 관계가 있다.





(a) parallel process (b) hierarchical process (c) synchronized process

그림 3. 표현 가능한 프로세스 종류.

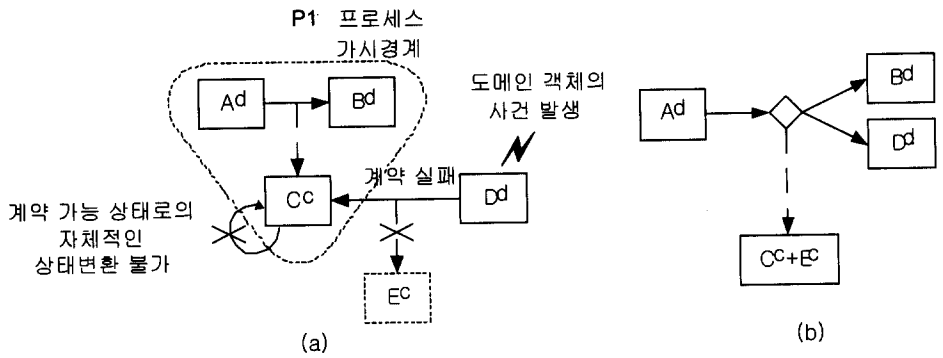


그림 4. 협동 프로세스의 막힘 현상과 해결 방안.

(3) 동기화 프로세스 : 계약-협동 네트가 세 개의 협동 프로세스 $P_1 = \{\{A \Rightarrow B\} \rightarrow C\}$, $P_2 = \{\{D \Rightarrow E\} \rightarrow F\}$, $P_3 = \{\{C, F \Rightarrow G\} \rightarrow H\}$로 형성된 경우 P_1의 계약 클래스 C와 P_2의 계약 클래스 F가 생성된 후 서버 클래스 G에게 계약을 의뢰한다. 따라서 프로세스 P_3는 P_1과 P_2의 협동 단계가 동기화될 때 시작된다(<그림 3(c)> 참조). 동기화 프로세스는 계층적 프로세스의 특수한 경우이다.

2.4 막힘 현상(Deadlock)

계약-협동 네트 모형을 구축하다 보면 프로세스가 더 이상 진행되지 않는 막힘 현상이 발생할 수 있다. 계약-협동 네트에서의 막힘 현상은 제약식을 더 이상 만족시키지 못할 때 발생하며, 클라이언트 객체의 삽입 취소 사건에 의한 막힘 현상 제거는 프로세스 진행에 의미가 없기 때문에 고려 대상에서 제외한다.

[정리 1] 협동 프로세스 $\{\{D^d \Rightarrow C^c\} \rightarrow E^c\}$, 즉 도메인 객체로부터 계약 객체로의 계약의존 관계를 맺는 프로세스는 막힘 현상을 발생시킨다.

(증명) <그림 4(a)>와 같이 클라이언트 객체인 도메인 객체의 사건 발생에 의해서 제약식이 파괴된 경우 서버 객체인 계약 개체는 사건을 일으켜서 제약식을 만족시켜야 한다. 그러나 계약 객체는 스스로 사건을 발생시킬 수 없기 때문에 계약이 더 이상 진전되지 않는다. 또한 [가정 2]에 의해서 계약 객체를 생성한 계약의존 관계에 있는 객체들은 도메인 객체를 인식하지 못하므로 이 객체들이 계약 객체의 상태변수를 갱신해서 도메인 객체와 계약을 맺도록 하지는 못한다.

[정리 1]에서 발생하는 막힘 현상을 협동 단계 막힘 현상(Collaboration Stage Deadlock)으로 정의한다. 협동 단계 막힘 현상은 주로 계층적 프로세스에서 협동 과정 중에 있는 상위 협동 프로세스의 계약 개체가 다른 도메인 객체로부터 계약의존 관계를 형성하는 경우에 발생한다. 협동 단계 막힘 현상을 해결하기 위해서는 <그림 4(b)>에서와 같이 세 개의 도메인 객체를 하나의 계약으로 묶는 다자간(N-ary) 계약의존 관계를 형성해야 한다.

[정리 1]로부터 다음과 같은 보조 정리가 성립된다.

[보조 정리 1] 계약 객체(클라이언트)로부터 계약 객체(서버)로의 계약의존 관계, 즉 $P = \{\{A^c \Rightarrow B^c\} \rightarrow C^c\}$는 막힘 현상을 유발한다.

(증명) 생략

<그림 4c>와 같이 계약 객체(C)로부터 계약 객체(F)로의 계약의존 관계는 협동 프로세스의 동기화에서 발생하는 데 협동 단계 막힘 현상을 방지하려면 계약의존 관계 설정시 도메인 객체(G)를 포함한 다자간 계약의존 관계를 맺어야 한다.

2.5 사건-조건-활동 규칙

제약식을 이용하여 선언된 비즈니스 메타 프로세스(계약-협동 네트에서는 협동 프로세스를 의미함)를 구현 가능한 비즈니스 프로세스로 전환하기 위해서는 사건-조건-활동 규칙을 적용한다. 사건-조건-활동 규칙은 객체의 삽입(생성), 삭제, 변경 등의 여러 사건에 대하여 앞 절에서 정의된 제약식을 파괴하는 지를 판단하여(조건) 상황에 맞는 비즈니스 로직을 수행(활동)하는 것을 표현하는 방법이다(Dayal *et al.*, 1995). 이러한 사건-조건-활동 규칙은 능동 데이터베이스에서 쉽게 구현될 수 있다는 장점이 있다.

고객으로부터 은행, 은행원으로의 계약의존 관계와 이로부터 상품을 참조하여 계좌가 정의되는 계약함수에 대한 협동 프로세스를 표현하는 제약식이 다음과 같을 때 이에 대한 사건-조건-활동 규칙을 생성한 결과가 <표 2>와 같다. 다음 제약식 중 계좌개설 관계의 표기 중 *의 의미는 특정 객체를 정하지 않음을 의미한다.

표 2. 사건 – 조건 – 활동 규칙의 예

제약식 구분	사 건	조 건	활 동
C_1의 첫번째 제약식	고객 생성	신용상태 양호하면	은행과 은행원 선택 거래신청 관계 시작
	고객 신용 상태 변경	신용상태 불량해지면	거래신청 관계 정지
C_1의 두번째 제약식	거래신청 관계 시작		계좌 생성
	거래신청 관계 정지		계좌 해지
C_2 제약식	계좌 상태 변경	계좌의 잔고가 출금한도액보다 적어지면	계좌 상태 변경 작업 취소(출금작업 취소)
	계좌 상태 변경	계좌의 잔고가 입금한도액보다 커지면	계좌 상태 변경 작업 취소(입금작업 취소)
C_3 제약식	계좌해지 신청시	계좌의 상태가 '잡좌'가 아니면	계좌개설 관계 정지, 계좌 삭제
	계좌해지 신청시	계좌의 상태가 '잡좌'이면	계좌해지 신청 취소

[C_1 제약식]

$$\forall c : \text{고객}, \text{거래신청}(c), \neg(c.\text{신용상태} = \text{'불량'}) \rightarrow \exists b : \text{은행}, \exists w : \text{은행원}, \text{거래신청 관계}(c, b, w)$$

$$\forall c : \text{고객}, \forall b : \text{은행}, \forall w : \text{은행원}, \forall g : \text{상품}, \text{거래신청 관계}(c, b, w) \rightarrow \exists a : \text{계좌}, \text{계좌개설 관계}(c, b, w, g, a)$$

[C_2 제약식]

$$\forall g : \text{상품}, \forall a : \text{계좌}, \text{계좌개설 관계}(*, *, *, g, a), (a.\text{잔고} \geq g.\text{출금한도액})$$

$$\forall g : \text{상품}, \forall a : \text{계좌}, \text{계좌개설 관계}(*, *, *, g, a), (a.\text{잔고} \leq g.\text{예금한도액})$$

[C_3 제약식]

$$\forall c : \text{고객}, \forall b : \text{은행}, \forall a : \text{계좌}, \text{계좌개설 관계}(c, b, *, *, a), \neg(a.\text{상태} = \text{'잡좌'}) \rightarrow \text{계좌해지 신청}(c, b, a)$$

제약식으로부터 사건 – 조건 – 활동 규칙을 생성하는 방법을 기술하면 다음과 같다. 제약식 파괴 가능한 사건을 추출하여야 하는데, 우선 ∀(For All)과 ∃(There Exist) 각각 →의 좌측항과 우측항에 배치된 경우에는 좌측항의 객체들의 삽입 사건과 우측항의 객체의 삭제 사건이 고려 대상이 된다. 여기에 제약식에 포함되어 있는 여러 항의 진리값을 변화시킬 수 있는 변경 사건들이 포함된다. 이와 같이 제약식 파괴 가능한 사건을 추출한 후 실제 제약식을 파괴하는 조건을 찾아낸다. 그리고 그 경우에 해당하는 비즈니스 로직을 활동 부분에 기입하면 된다.

계약 – 협동 네트의 각 협동 프로세스별로 생성된 사건 – 조건 – 활동 규칙들은 실제 비즈니스 프로세스 집합이 된다. 이러한 프로세스들의 활동에 의해서 또 다른 추가, 삭제, 변경 사건이 발생하며 이러한 사건들이 다른 사건 – 조건 – 활동 규칙을 활성화하여 복잡한 비즈니스 프로세스를 구성한다.

3. 기존 연구 고찰 및 문제점

기존의 비즈니스 프로세스 분석 방법은 엄격한 분석 기반이 없는 비정형(Informal), 절차적(Procedural) 방법론이 대부분이다. 이런 방법들은 사건과 이를 처리하기 위한 활동(Activity)들의 선행 관계를 비즈니스 프로세스로 정의한다. 절차적 프로세스 분석 방법은 다시 다음과 같이 세 종류로 구분해 볼 수 있다.

- 활동지향 방법론 : 구조적 또는 함수 지향적 방법론이라고도 하며, 활동과 입출력 데이터를 바탕으로 활동 간의 선행 관계를 분석하는 방법론이다. 대표적인 예는 SADT(Ross and Shoman, 1978) 및 이를 이용한 IDEF0 방법론(Bravoco and Yadav, 1985; KBSI)이 존재한다. 이들 방법론은 활동의 재귀적 분할을 통하여 복잡한 프로세스를 Divide-and-Conquer 방식으로 분석한다.
- 상태지향 방법론 : 여기서 상태란 일반적으로 활동의 결과를 뜻하며, 상태 중심 방법론은 상태를 중심으로 의사결정을 행하여 프로세스의 분지를 가능하게 한다. 이 방법론은 활동의 병렬 처리, 동기화 및 조건부 분기 등을 명확히 표현할 수 있는 장점이 있다. 대표적인 예는 Petri Net (Murata, 1989)과 State Chart(Harel and Gery, 1997)가 있다. 그러나 상태 중심 방법론도 활동의 결과를 상태로 표현하기 때문에 표현 형식만 다를 뿐이지 활동 중심 방법론과 같이 절차적 방법의 일종이다.
- 객체지향 방법론 : 객체의 상태변수와 메쏘드로 구성되어 있다. 객체 중심 방법론에서는 프로세스를 객체들 간의 메쏘드 호출 순서로 표현한다. 대표적인 객체 중심 방법론으로는 UML에서 제시하는 Use Case 접근 방법이 있다 (Jacobson, 1992; Larman, 1998). 그러나 Use Case접근 방법은 정형화된 프로세스 생성 규칙을 제시하지 못하기 때문에

동일 시스템일지라도 서로 다른 Use Cases를 도출하는 경우가 많다. 이러한 원인은 UML이 객체 중심 방법론이라 할지라도 객체를 먼저 분석하지 않고 활동 중심 방법론과 유사한 Use Case 분석부터 시작하기 때문이다. 따라서 Use Case 분석은 분석자에 따라서 분석결과가 달라지고 결국 클래스도 다르게 정의될 수 있는 문제점이 존재한다. 이밖에 Use Case 방법을 컴포넌트 개발 관점에서 수정하여 적용한 Catalysis 방법이 있다(D'souza and Wilis, 1999).

비즈니스 프로세스를 위와 같은 절차적 방법으로 분석하고 표현하기에는 다음과 같은 단점이 있다.

첫째, 프로세스 단위의 불명확성 : 절차적 방법들은 프로세스의 구성요소인 활동의 분할(Decomposition)을 허용하며 분할의 일정한 기준이 없다. 따라서 분석자마다 활동의 분할 기준이 달라지기 쉽기 때문에 동일 시스템의 프로세스를 분석하더라도 서로 다른 프로세스 분석 결과를 얻는 경우가 많다.

둘째, 객체 표현의 한계 : 절차적 방법들은 활동 중심으로 비즈니스 프로세스를 표현하기 때문에 활동의 주체(Who)가 되는 수행 객체를 명확히 표현하지 못한다. 객체지향 프로세스 분석 방법인 UML의 Use Case 방법도 시스템의 객체를 추출한 후 프로세스를 표현하지 않는다. 다시 말해서 Use Case 방법은 우선 활동들의 선행 관계를 표시한 후 설계 단계에서 객체들의 호출 관계로 프로세스를 표현한다. 따라서 첫 번째 문제점인 프로세스 단위의 불명확성으로 인하여 객체들의 호출 관계도 달라질 수 있다.

셋째, 사건의 연관성 : 사건은 객체에 의해서 발생하며 프로세스는 수행 객체들의 활동(메쏘드)에 의해서 진행된다. 프로세스 진행 중에 이들 수행 객체에서도 사건이 발생할 수 있으며, 따라서 한 사건에 의해 진행되는 프로세스가 수행 객체들의 사건에 의해서 새로운 프로세스를 유발시킨다. 이러한 사건 간의 연관성은 프로세스의 단위, 특히 프로세스 종료 시점의 정의를 어렵게 만드는 요인이다.

넷째, 협동 프로세스 표현 한계 : 비즈니스 도메인에서는 CSCW(Computer Supported Collaborative Work)와 같이 객체들 간의 호출 순서가 명확히 정의되지 않는 프로세스가 존재하며 이를 협동(Collaboration) 프로세스라고 한다. 이런 프로세스는 객체 상호 간 합의에 도달할 때까지 반복적인 호출이 발생하며, 이를 해결하기 위해서 일반적으로 조정자(Mediator)를 두어서 합의를 도출하게 만든다. 기존의 절차적 방법으로는 이러한 협동 프로세스를 표현하기 매우 어렵다.

이밖에 계약-협동 네트에 관련된 연구는 다음과 같다. 계약-협동 네트에서 제약식을 도입한 아이디어는 Predicate-Transition Net(Genrich and Lautenbach, 1981)이라는 Petri Net과 유사하나, Predicate-Transition Net에서는 계약-협동 관계가 존재하지 않으며 또한 객체지향 방법도 아니다. 제약식을 데이터베이스 스키마 정의에 도입한 연구로는 Urban and Lois(1990)이 있으며, 최근에 UML에 제약식을 표현하는 방법이 제시되기도 하였다(Warner and Kleppe, 1999). 그러나 UML에서 제시한 것은 단지 제약식을 표현하는 언어에 불과하다. Kent(1997)는 객체의 상태 변수 간의 제약식을 다이어그램 형태로 표현하였으나, 이 또한 계약-협동 관계를 이용해서 프로세스를 정의해 나가는 방식은 아니다.

한편 워크플로우 모델링에 관한 연구[1]로는 Van Der Aalst *et al.* (1994), Kim and Nof (2000), Medina-Mora *et al.* (1993) 등이 있으며, Lei and Singh (1997)은 워크플로우 모델링 기법을 정리하여 소개하였다. 그러나 대부분의 기법은 앞에서 언급한 절차적 방법을 따른다. 이 밖에 Scheer (1999)는 사건의 연관성 및 사건과 함수의 관계를 메타 수준에서 정의하였다.

4. 결 론

본 연구에서는 선언적 메타 프로세스 기술 방법인 계약-협동 네트를 제시하였다. 계약-협동 네트는 계약의존 관계와 계약함수에 의해서 관련된 객체들의 상태변수들이 만족해야할 제약식을 기술하고 이 제약식들이 파괴되는 것을 사건으로 보며, 제약식을 다시 만족하기 위해서 객체들 간의 호출을 프로세스로 규정한다. 프로세스는 사건-조건-활동 규칙을 이용하여 묘사할 수 있음을 보였다.

추후 연구 과제로는 첫째, 계약-협동 네트를 Process Algebra(Baeten and Weijland, 1990)나 Computational Tree Logic(Clarke *et al.*, 1986)을 이용하여 좀 더 정형화하는 연구와 둘째, 계약-협동 네트를 바로 구현할 수 있는 워크플로우 엔진 구현에 관한 연구와 셋째, 각 객체를 에이전트로 보고 다중 에인전트 기술을 이용하여 파괴된 제약식을 원상 복귀하는 CSP(Constraint Satisfaction Problem)에 대한 연구 등이 있다. 또한 본 연구가 프로세스 모델링에 초점을 맞추었기 때문에 고려되지 않은 객체지향 모형의 정적인 측면을 통합하여, 일관성 있는 모델링 방법론을 구축하는 것이 본 연구의 궁극적인 목표이다.

참고문헌

Baeten, J. C. M. and Weijland, W. P. (1990), *Process Algebra*, Cambridge University Press.

Bravoco, R. R. and Yadav, S. B. (1985), A methodology to model the functional structure of an organization, *Computers in Industry*, **6**, 345-361.

Clarke, E. M., Emerson, E. A. and Sistla, A. P. (1986), Automatic verification of finite-state concurrent systems using temporal logic specification, *ACM Transactions on Programming Languages and Systems*, **8**(2), 244-263.

1) 본 연구에서는 워크플로우 시스템에 관한 기존 연구는 참조하지 않음

Dayal, U., Hanson, E. and Widom, J. (1995), Active database systems, Modern Database Systems (Ed. Kim, W.), Addison-Wesley.

DeMarco, T. (1979), *Structured Analysis and System Specification*, Yourdon Press, Englewood Cliffs.

D'Souza, D. F. and Wilis, A. C. (1999), *Objects, Components, and Frameworks with UML: The Catalysis Approach*, Addison-Wesley.

Genrich, H. J. and Lautenbach, K. (1981), System modeling with high-level Petri nets, *Theoretical Computer Science*, **13**, 109-136.

Harel, D. and Gery, E. (1997), Executable object modeling with state charts, *IEEE Computer*, July, 31-42.

Jacobson, I. (1992), *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley.

KBSI, White papers on IDEF series, Available at http://idef.com.

Kent, S. (1997), Constraint diagrams: visualizing invariants in object oriented models, *Proceedings of OOPSLA97*, ACM Press.

Kim, C.-O. and Nof, S. Y. (2001), A workflow system for coordinating and integrating distributed CIM information systems, Accepted for Publication, *IEEE Transaction on Systems, Man, and Cybernetics.*

Larman, G. (1998), *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*, Prentice-Hall.

Lei, Y. and Singh, M. P. (1997), A comparison of workflow metamodels, *Proceedings of the ER-97 Workshop on Behavioral Modeling and Design Transformations: Issues and Opportunities in Conceptual Modeling*, Los Angeles, CA.

Maier, D. and Ullman, J. D. (1983), Maximal objects and the semantics of universal relational databases, *ACM Transactions on Database Systems*, **8**(1), 1-14.

Medina-Mora R., Wong, H. K. T. and Flores, P. (1993), Action Workflow as the enterprise integration technology, *IEEE Data Engineering Bulletin*, **16**(2), 49-52.

Murata, T. (1989), Petri nets: properties, analysis, and applications, *Proceedings of the IEEE*, **77**(4), 541-580.

Ross, D. and Shoman, K. (1978), Structured analysis for requirement definition, *IEEE Transaction on Software Engineering*, **3**(1).

Scheer, A-W. (1999), *ARIS - Business Process Modeling*, Springer.

Taylor, D. A. (1995), *Business Engineering with Object Technology*, John Wiley and Sons.

Urban, S. D. and Lois. M. L. (1990), Constraint analysis: a design process for specifying operations on objects, *IEEE Transactions on Knowledge and Data Engineering*, **2**(4), 391-400.

Van Der Aalst, W. M. P., Van Hee, K. M. and Houben, G. J. (1994), Modelling workflow management systems with high-level Petri nets, *Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, 31-50.

Warmer, J. B. and Kleppe, A. G. (1999), *The Object Constraint Language : Precise Modeling With UML*, Addison-Wesley.