UML 2.0을 지원하기 위한 ebXML BPSS의 개선

Improving ebXML BPSS based on UML 2.0

김자희(Ja-Hee Kim)*

초 록

ebXML은 비즈니스 프로세스를 UMM이란 방법론으로 모델링하고 BPSS로 구현한다. 현재의 UMM과 BPSS는 UML 1.4를 기반으로 개발되었다. 그러나 비즈니스 프로세스 모델링에 좀 더 적합하도록 변경된 UML 2.0을 UMM에 도입할 경우 표현력과 유지보수의 효율성 등은 향상되겠지만 일부 기능은 BPSS로 구현이 불가능해 진다. 본 논문에서는 먼저 UML 2.0의 새로 도입된 요소들 중에서 ebXML을 향상시키는 데 도움을 줄 수 있는 개념들을 소개한다. 그리고 그 개념들이 UMM에 도입되었을 때 이를 구현할 수 있도록 BPSS를 보완하는 방법을 제안한다.

ABSTRACT

In ebXML the choreography of a business process should be modeled by UMM and is finally realized in BPSS. The current UMM and BPSS are developed based on UML 1.4. Recently OMG introduces the UML 2.0, which strengthens the modeling power of activity diagrams and components. If we adopt UML 2.0, the modeling power of UMM is improved but BPSS cannot implement the modeling features. In this paper, we examine the new features of the activity diagram in UML 2.0. We also propose ways to improve the UMM and the BPSS using the new features.

키워드 : 비즈니스 프로세스, 전자상거래 표준
ebXML, UMM, BPSS, UML

* 서울산업대학교 IT정책전문대학원 전임강사

1. 서 론

ebXML (electronic business XML)은 UN/CEFACT (United Nations/ Centre for Trade Facilitation and Electronic Business)와 OASIS (Organization for the Advancement of Structured Information Standards)에서 제안한 국제 전자거래 표준으로 2004년 ISO 15000표준으로 승인되었다[3]. ebXML에서는 프로세스의 제어 흐름인 코레오그라피 (Choreography)를 UMM (UN/CEFACT Modeling Methodology)으로 모델링하도록 권고하고 있다. UMM의 결과물들은 현재 UML (Unified Modeling Language) 1.4의 다이어그램 형태로 표현되며 최종적으로 BPSS로 구현된다. BPSS는 처음에는 UN/CEFACT TMG의 주관으로 만들어져 1.1까지 만들어졌으나 OASIS에서는 ebBP라는 이름으로 독자적으로 BPMN을 기반으로 ebXML을 위한 비즈니스 프로세스 구현 언어를 개발하여 현재 2.04까지 발표하였다[13]. 이 두 단체의 BPSS 표준의 불일치를 해결하기 위하여 2006년 3월 OASIS의 차세대 BPSS를 *ISO/DTS* 15000-6의 표준으로 신청하기로 합의하고 BPSS 3.0부터는 두 단체가 협력하여 개발하기로 협력하였다[5]. 본 논문에서는 UN/CEFACT에서 개발한 1.1을 기준으로 설명하되 ebBP 2.0.3을 추가적으로 언급한다. 가장 최근 발표된 UML 2.0은 실시간 시스템 (Real Time System)과 워크플로우 (Workflow) 시스템을 모형화하기 위해 다양한 개념 및 요소들을 도입하였고 CBD (Component Based Development)를 지원하고 있다[14]. 이를 위하여 UML 2.0에서는 사용도 (Use Case)와 작업도 (Activity Diagram) 등에 많은 변화를 주었다[12]. 이 중에서도 UMM과 BPSS의 향상에 가장 많은 기여를 할 것으로 기대되는 것은 코레오그라피를 모델링하는 데 사용되는 작업도다.

작업도의 변화 중 가장 두드러지는 것은 동시성 시스템 (Concurrent System)의 모델링표현력을 향상시키기 위하여 기존의 상태도 (State Chart)기반의 작업도를 페트리넷 (Petri Net) 기반으로 바꾸었다는 것이다. 이러한 개념의 변화는 일부 용어를 변화시켰다. 예를 들어 상태 (State)와 전이 (Transition)는 노드 (Node)와 에지 (Edge)로 각각 변화되었다. 그러나 더 근본적인 변화는 토큰 (Token)의 도입이다. 토큰은 객체를 표현할 수도 있고 그 외에 자료나 제어의 현재 위치를 표현할 수도 있다. 예를 들어 중앙버퍼노드 (Central Buffer Node)와 자료는 객체 토큰을 저장할 수 있고 다중 실행에서는 여러 개의 토큰을 하나의 장소에 저장함으로써 동시에 진행하는 시스템을 모델링하는 능력을 강화시킨다.

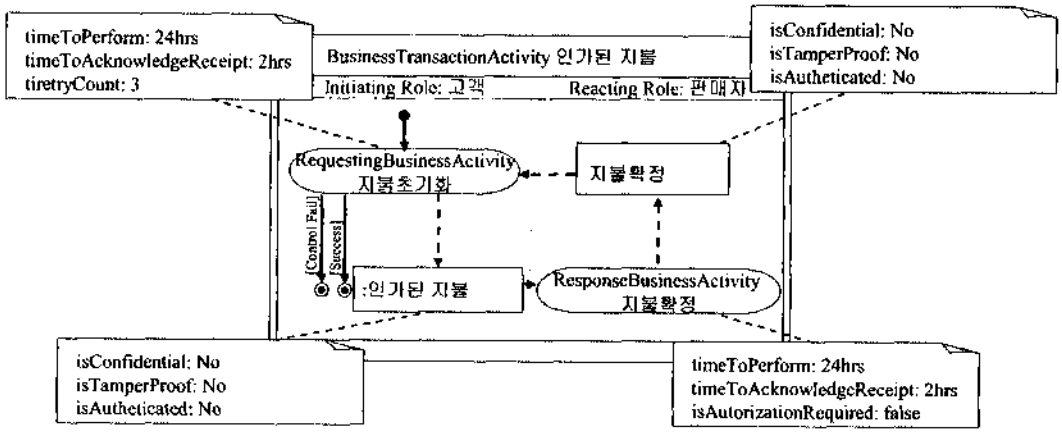
UML 2.0으로 바뀌면서 UML의 표현력도 향상되었다[7,10,18]. 비즈니스 프로세스의 표현력은 Aalst 등이 분류한 워크플로우 패턴에 의해 측정, 비교 될 수 있다[4,17]. Aalst 등은 상용 워크플로우 관리 시스템을 분석하여 20가지 워크플로우 시스템을 도출했다. UML 2.0을 사용할 경우 UMM은 좀 더 풍부한 표현력을 가질 수 있을 것으로 기대된다[10]. 그러나 이 경우 UMM을 통해 모델링된 비즈니스 프로세스 중 일부는 BPSS로 구현할 수 없다. 그러므로 본 논문에서는 UML 2.0 기반의 UMM을 통해 모델링 된 비즈니스 프로세스를 BPSS로 구현할 수 있도록 BPSS를 개선시키는 방법을 제안한다.

UML 2.0의 또 다른 변화는 작업 (Activity)에 수준 (Level)이라는 개념을 도입한 것이다. UML은 시간이 지날수록 다양한 개념을 포함하고 표현력이 증가되는 반면에 점점 복잡해 졌다. 이로 인해 UML을 이용하는 사용자들의 대부분은 UML의 전체 기능을 사용하기 보다는 자신들의 목적에 따라 UML의 일부 기능만을 사용하고 있다. 작업의 수준이란 이런 사용자들을 위하여 사용자의 목적

에 따라 필요한 기능을 6단계의 수준으로 나누어 필요한 개념들을 정리해 놓은 것이다. 이 외에도 UML 2.0의 작업도는 모델을 좀 더 간략하면서도 이해하기 쉽게 모델링할 수 있도록 인수 집합(Parameter Set) 등의 다양한 개념을 소개하고 있다. 본 논문에서는 이 새로운 개념들을 자세히 설명하고 이러한 개념들이 UMM에 도입되었을 때 이를 BPSS로 구현할 수 있는 방법을 예제와 함께 설명하고자 한다. 2장에서는 UMM과 BPSS에 대해 간략하게 살펴보고 3장에서는 워크플로우 패턴에 기반한 BPSS의 표현력을 향상시키는 방법에 대하여 설명하고자 한다. 그리고 4장과 5장에서는 작업의 수준과 인수 집합들의 개념을 살펴보고 UMM에서 이 개념들을 도입하였을 때 BPSS를 어떤 식으로 변경하여야 하는지를 다룬다. 그리고 6장에서는 제안된 새로운 BPSS에 대하여 정리한다.

2. UMM과 BPSS

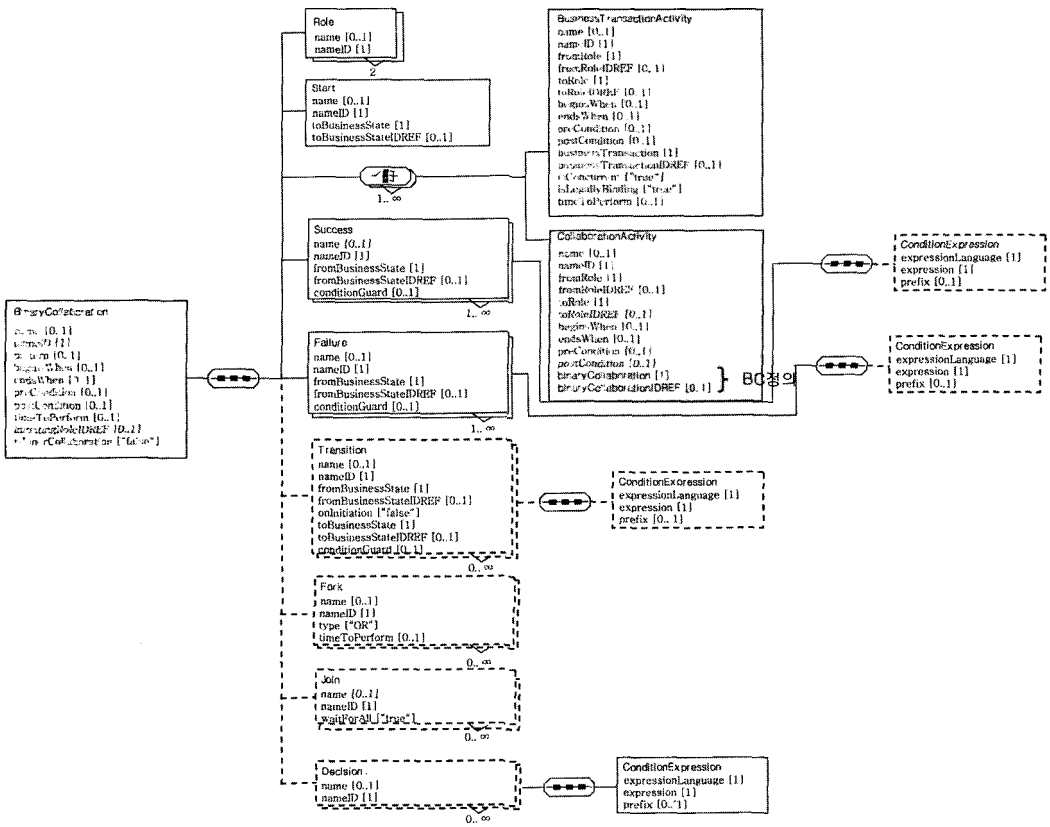
ebXML에서는 비즈니스 프로세스를 모델링하기 위하여 UMM이라는 모델링 방법론의 사용을 권고하고 있다[9]. UMM은 Open-EDI의 참조모델에서 비즈니스 결정과 조직들 간의 확약(Commitment) 들에 중점을 둔 BOV (Business Operational View)를 중심으로 한 모델링 방법론을 제시하고 있다. UML로 기술되는 UMM은 기술 독립적이므로 B2B에서의 거래를 구현할 때 웹서비스와 BPSS 등을 모두 허용한다. UML에 기반을 둔 UMM은 조직 간의 비즈니스 프로세스에서 비즈니스적인 면을 모델링하기 위한 UML 프로파일(Profile)을 정의한다. UMM 방법론은 BDV (Business Domain View)와 BRV (Business Requirements View), BTV (Business Transaction View), BSV (Business Service View)라는 4 가지 관점을 가지고 있다. BDV는 현재의 비즈니스 프로세스 상황을 파악하는 데 사용된다. 즉, BDV는 현재 비즈니스 프로세스에 관한 AS-IS 지식을 모으지만 새로운 비즈니스 프로세스를 생성하지는 않는다. BRV는 고려하고 있는 협력 (Collaboration) 중에서 가능한 비즈니스 협력을 파악하고 이 협력들의 요구사항을 상세히 정의한다. BTV는 비즈니스 협력을 위한 코레오그라피와 교환되는 비즈니스 정보의 구조를 정의한다. 마지막으로 BSV은 기본적으로 네트워크 구성요소들 간의 상호작용을 묘사한다.



〈그림 1〉 BT의 예

본 논문은 비즈니스 협력이라 불리는 조직 간의 비즈니스 프로세스들의 코레오그라피 관점에서 UMM과 BPSS을 분석하고 개선방안을 모색하므로 BTV를 위주로 설명한다. 비즈니스 협력은 협력에 관여하는 관계자의 수에 따라 양자간 협력 혹은 다자간 협력 (Multi-party Collaboration)으로 분류할 수 있다. BPSS 1.1은 양자간 협력인 BC (Binary Collaboration) 요소를 기본으로 하며 ebBP는 일반적인 다자간 협력인 Business Collaboration을 기본 요소로 한다. 비즈니스 협력은 일반적으로 다수의 비즈니스 파트너들 간의 작업들이 복잡하게 구성되어 있지만, 가장 기본적인 비즈니스 협력은 한 쪽에서 협력을 초기화하여 상대 쪽으로 요구를 보내고 상대편에서는 요구를 받고 경우에 따라 그에 대한 반응을 되돌려 주는 BC이다. 이 가장 간단한 비즈니스 협력인 BC가 모든 작업의 기본단위로, 실패할 경우 이전 상태로 되돌려야 하는 일종의 트랜잭션이다. ebXML에서는 이런 기본단위를 6가지 패턴으로 분류하고 BT (Business Transaction)라고 부르며 <그림 1>과 같은 작업도로 표현한다. <그림 1>은 UMM의 "인가된 지불"의 예를 BT로 모델링한 모습을 보여주고 있다.

그러므로 BT는 언제나 2명의 파트너와 2개의 작업으로 이루어져있다. 각 작업은 비즈니스 파트너 중 하나에 의해 수행된다. 첫 번째 작업은 상대 비즈니스 작업을 초기화하기 위한 정보를 보내는 것이다. BT가 간단한 정보 전달이나 통지를 위한 것일 경우 트랜잭션이 종료된다. 그러나 답변이 필



<그림 2> BPSS 1.1

요한 경우라면 상대 비즈니스 작업은 초기 비즈니스 작업에게 비즈니스 정보를 보낸다. <그림 1>에서도 고객이라는 파트너가 "지불초기화"라는 작업을 통해 판매자에게 "인가된 지불"을 보낸다. 이 경우 판매자는 지불이 성공적인지를 고객에게 알려줄 필요가 있다. 그러므로 초기 비즈니스 작업인 "지불초기화"에게 지불이 확정되었음을 알리기 위한 문서를 보내다.

일반적인 비즈니스 협력은 복수 개의 작업들로 이루어져 있기 때문에 UMM에서는 작업들 간의 실행순서를 정의하는 코레오그라피가 중요하다. 코레오그라피 내의 각 작업은 다른 비즈니스의 협력이나 BT로 상세 정의될 수 있다. 단, 비즈니스 협력 안에 정의된 작업들 중에는 자기 자신을 포함하는 재귀적 정의는 불가능하다. UMM의 다자간의 거래를 모델링하는 비즈니스 협력 규약과 양자 간의 기본 협력단위 만을 모델링하는 BT 모두 UML 1.4의 작업도를 기반으로 하고 있다. 그러므로 UMM은 UML 1.4의 모델링 능력과 함께 모든 한계를 갖고 있다. 우리는 이러한 한계를 UML 2.0을 이용하여 어떻게 해결할 것인지에 대하여 논의할 것이다. ebBP는 BPMN을 기반으로 하나 비슷한 요소들을 가지고 있다. 그러나 어떤 언어를 기반으로 하고 있는지에 관계없이 비즈니스 프로세스의 모델링을 할 때에 UMM의 사용이 권고되므로 BPSS의 표현력은 UMM보다 풍부해야 UMM의 결과물을 모두 실현할 수 있다. BPSS의 최종 목적은 e-비즈니스 프로세스 모델과 e-비즈니스 소프트웨어 요소 정의 간의 간극을 메우는 것이다[16]. BPSS는 비즈니스 파트너들 간의 협력에 관한 코레오그라피들을 정의할 수 있도록 XML 스키마를 제공한다. 그리고 e-비즈니스 소프트웨어 요소 집합들 간의 협력을 실행시키기 위한 실시간 시스템의 구성 인수들을 제공한다. 이와 관련된 모든 요소들은 <그림 2>와 같이 XML 스키마로 정의된다. BPSS에서 코레오그라피는 <그림 2>와 같은 BC (Binary Collaboration) 요소로 정의된다. BC의 각 작업들은 자식요소인 BusinessTransactionActivity와 CollaborationActivity 요소로 정의되고 나머지 BC의 자식요소들은 이 작업들 간의 순서 제어와 관련이 있다. 그리고 CollaborationActivity 내부에 다시 BC를 정의할 수 있는 binaryCollaboration과 binaryCollaborationIDREF가 존재하므로 계층적인 비즈니스 프로세스 정의가 가능하다. ebBP의 경우 코레오그라피에 관한 부분은 collaborationGroup으로 묶여있다.

3. 워크플로우 패턴 기반의 분석

워크플로우 분야에서는 다양한 프로세스의 순서와 흐름, 통제 패턴 등에 따라 워크플로우 패턴을 분류하는 연구가 수행되어 왔다. 현재 가장 많이 사용되고 있는 워크플로우 패턴은 Aalst 등이 상용 워크플로우 관리 시스템을 분석하여 정리한 20가지 워크플로우 패턴이다[17]. 이 워크플로우 패턴을 이용하면 다양한 종류의 비즈니스 프로세스 언어들 간의 표현력을 분석하고 이들 간의 변환 방법론을 만들 수가 있다[4,8,10,11,18]. Kim과 Huemer는 UML 1.4를 기반으로 한 UMM과 이 UMM을 구현하는 BPSS에서 지원하는 워크플로우 패턴을 분석하고 변환 방법론을 제시하였다[10]. 그러나 일부 워크플로우 패턴은 UMM이 지원함에도 불구하고 BPSS에서는 지원하지 않기 때문에 BPSS의 변경이 요구되었다. 게다가 UML 2.0은 UMM의 표현력을 향상시켜 BPSS가 구현할 수 없는 UMM의 워크플로우 패턴 수를 증가시킨다[1,2,8,14]. Kim과 Huemer는 UML 2.0을 도입할 때 UMM이 추가적으로 지원할 수 있는 워크플로우 패턴을 <표 1>과 같이 정리하였다. 표에서 '+'는

〈표 1〉 UMM과 BPSS가 지원하는 워크플로우 패턴

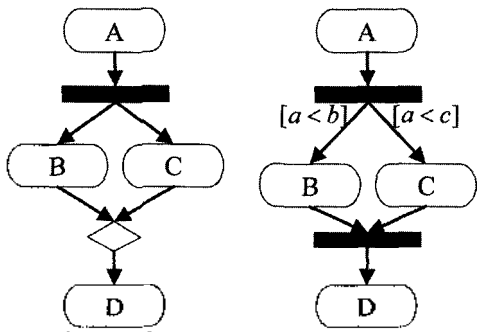
	UMM	BPSS
순차 (Sequence)	+	+
병렬분기 (Parallel Split)	+	+
동기화 (Synchronization)	+	+
배타분기 (Exclusive Choice)	+	+
단순결합 (Simple Merge)	+	+
복수선택 (Multi Choice)	+	+
동기결합 (Synchronizing Merge)	+	-
복수결합 (Multi Merge)	2	-
판별 (Discriminator)	2	+
임의순환 (Arbitrary Cycle)	2	+
암묵적 종료 (Implicit Termination)	+	+
비동기 다중 인스턴스 (MI without Synchronization)	+	+
설계 중 복수 인스턴스 (MI with a Priori Design Time Knowledge)	-	-
실행 중 복수 인스턴스 (MI with a Priori RunTime Knowledge)	-	-
임의 복수 인스턴스 (MI without a Priori RunTime Knowledge)	-	-
지연분기 (Deferred Choice)	+	+
임의 라우팅 (Interleaved Parallel Routing)	-	-
마일스톤 (Milestone)	+	+
취소 작업 (Cancel Activity)	-	-
프로세스 취소 (Cancel Case)	+	+

UML 1.4에서도 이미 지원하는 워크플로우 패턴이고 아직 지원하지 않을 경우에는 '-'로 표기하였다. UML 2.0기반의 UMM이 추가적으로 지원하는 워크플로우 패턴은 '2'로 표기하였다.

BPSS는 UMM에 의해 모델링 된 비즈니스 프로세스를 구현할 수 있어야 하므로 UMM으로 모델링 된 모든 워크플로우를 구현할 수 있어야 한다. 그러나 〈표 1〉의 워크플로우 패턴 중에 이중 선 안에 있는 동기결합과 복수결합은 BPSS로 구현이 불가능하므로 실제 BPSS를 이용하여 구현할 때에 문제를 야기 시킬 수 있다. 이를 해결하기 위해서 아래 합류 (Join) 요소에 [008]줄과 같이 합류 요소에 제어가 도착할 때마다 다음 작업을 진행시킬 것인지를 결정하는 속성 (Attribute)을 추가할 것을 제안한다.

```
[001] <xsd:element name="Join">
[002]   <xsd:complexType>
[003]     <xsd:sequence>
[004]       <xsd:element ref="Documentation" min
            Occurs="0" maxOccurs="unbounded" />
[005]     </xsd:sequence>
[006]     <xsd:attributeGroup ref="name" />
```

```
[007]   〈xsd:attribute name = "waitForAll"
        type = "xsd:boolean" default = "true" /〉
[008]   〈xsd:attribute name = "everyTime"
        type = "xsd:boolean" default = "false" /〉
[009]  〈/xsd:complexType〉
[010] 〈xsd:unique name = "Join-ID"〉
[011]   〈xsd:selector xpath = "." /〉
[012]   〈xsd:field xpath = "nameID" /〉
[013] 〈/xsd:unique〉
[014] 〈/xsd:element〉
```



(a) 복수결합 (b)복수선택과 동기결합

〈그림 3〉 EPSS가 지원하지 않는 워크플로우 패턴

본 논문에서는 BPSS에서 지원해야 하는 동기결합과 복수결합 패턴의 의미와 해결방법을 〈그림 3〉의 예를 이용하여 설명한다. 복수결합은 〈그림 3a〉와 같이 병렬분기에 의해 나눠진 쓰레드(Thread) B, C가 끝날 때마다 다음 작업인 D를 수행하는 패턴이다. 즉, B가 C보다 먼저 끝날 경우, 작업은 "A 종료 → B, C 시작 → B 종료 → D 시작 → C 종료 → D 시작"과 같은 순으로 진행되게 된다. 이를 BPSS 코드로 단순화하려면 다음과 같이 합류 요소에 제어가 도착할 때 마다 다음 작업을 시작할 수 있도록 everyTime의 값을 false로 지정해 주어야 한다.

```
[001] 〈BusinessTransactionActivity name = "A"/〉
[002] 〈BusinessTransactionActivity name = "B"/〉
[003] 〈BusinessTransactionActivity name = "C"/〉
[004] 〈BusinessTransactionActivity name = "D"/〉
[005] 〈Transition fromBusinessState = "A"
      toBusinessState = "parallel split" /〉
[006] 〈Transition fromBusinessState = "parallel split"
      toBusinessState = "B" /〉
[007] 〈Transition fromBusinessState = "parallel split"
      toBusinessState = "C" /〉
[008] 〈Transition fromBusinessState = "B"
      toBusinessState = "multi merge" /〉
[009] 〈Transition fromBusinessState = "C"
      toBusinessState = "multi merge" /〉
[010] 〈Transition fromBusinessState = "multi merge"
      toBusinessState = "D" /〉
[011] 〈Fork name = "parallel split" type = "OR" /〉
[012] 〈Join name = "multi merge"
      waitForAll = "false" everyTime = "true" /〉
```

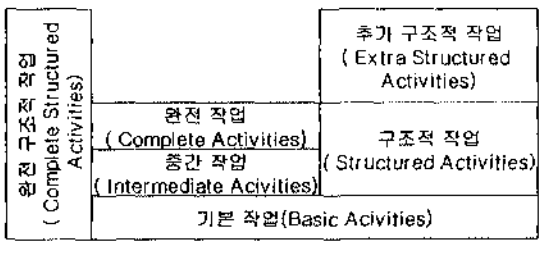
동기결합이란 〈그림 3b〉와 같이 복수선택에 의해 여러 개로 분기된 모든 쓰레드가 끝났을 때까지 기다리는 것이다. 이를 BPSS로 구현하기 위해서는 일반 동기화 패턴과 같이 합류 요소를 〈Join name = "synchronizing merge" waitForAll = "true" everyTime = "false" /〉로 정의하면 된다. 이와 같이 동기결합 패턴은 BPSS로 구현하기 쉽지만 이 BPSS를 실행시키는 실행 엔진에게 많은 무리를 주게 된다. 왜냐하면 〈그림 3b〉에서 실행하는 시점에서 B와 C가 모두 실행이 되었는지 아니면 B만 실행되었는지, 아무 작업도 실행되지 않았는지를 판단을 해서 합류요소로 제어가 도착했을 때 작업 진행여부를 결정해야 하기 때문이다. Aalst 등의 연구에 의하면 현재 많은 상용 워크플로우 관리 시스템이 실행 엔진의 구현 난이도 때문에 이 패턴을 제공하지 않고 있다고 한다[17]. ebBP에서는 이 문제를 해결하기 위하여 Fork 요소에 TimeToPerform이라는 속성을 추가하였다. 만일

Fork에 TimeToPerform의 값이 0보다 클 경우에는 작업의 콘트롤이 짝을 이루고 있는 Join으로 이동하게 된다.

본 장에서는 UML 20을 도입한 UMM이 지원하는 모든 워크플로우 패턴을 지원하기 위해 BPSS에 everyTime이라는 속성을 도입하였다. 그리고 이 속성을 이용하면 기존에 UMM에서만 지원했던 동기결합과 UML 20을 도입하면서 문제가 될 수 있었던 동기화 패턴 문제를 해결할 수 있음을 예로 보였다.

4. 작업의 6 단계 수준

이 장에서는 UML의 "작업 수준"이라는 관점에서 분석한다. UML 표준 전체는 너무 방대하고 복잡하기 때문에 대부분의 UML 사용자들은 자신들의 모델링 목적에 따라 UML의 일부만을 이용하여 모델링한다. 그래서 UML 20에서는 작업도 요소들을 6단계의 수준으로 분류하여 모델링의 목적에 따라 UML을 학습하고 사용할 수 있도록 도와주고 있다. 6단계의 작업 수준이란 기본 작업 수준 (Basic Activity Level)과 중간 작업 수준 (Intermediate Activity Level), 완전 작업 수준 (Complete Activity Level), 구조적 작업 수준 (Structured Activity Level), 추가 구조적 수준 (Extra Structured Level), 완전 구조적 수준 (Complete Structured Level)이다. 이 장에서는 각 수준들의 정의와 관계를 분석하고 UMM과 BPSS가 현재 지원하는 작업 수준과 지원 방법을 알아본다. 그리고 현재 지원하지 않는 개념 중 UMM과 BPSS가 지원해야 할 요소들을 살펴본다.

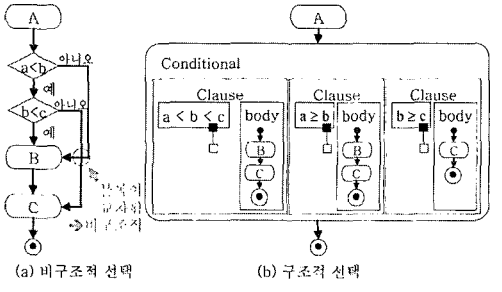


〈그림 4〉 작업의 6단계 수준의 관계

이 수준들 사이의 관계는 〈그림 4〉와 같이 정리될 수 있다. 〈그림 4〉에서 색으로 강조되어 있는 영역은 UMM과 BPSS에서 부분적으로 지원하는 수준이다. 이 중에서 가장 기본이 되는 수준은 "기본 작업 수준"이다. 이 수준은 전통적인 플로 차트 (Flow Chart)와 유사한 개념들을 지원한다. 이 수준에서는 UMM 전이 (Transition)와 초기 (Initial), 최종 (Final), 결정 (Decision), 병합 (Merge) 등의 유사 상태 (pseudo-State)들을 제공한다. BPSS에서 이 수준에 속한 요소는 각각 전이 (Transition), 비즈니스 트랜잭션 작업 (Business Transaction Activity), 협력 작업 (Collaboration Activity), 시작 (Start), 성공 (Success), 실패 (Failure), 결정 (Decision)이 있다. 기본 작업 수준에서는 구조적 제약을 가지고 있지 않다.

중간 작업 수준은 기본 작업 수준에 기반 한다. 이 수준의 표현력은 큐 (Queue)를 갖는 페트리 넷과 유사하다. 페트리 넷은 동시성이 있는 시스템을 그래프와 수학적 프레임워크를 가지고 표현할 수 있도록 도와주는 정형화된 방법 (Formal Method) 중 하나다[12]. 이 중간 작업 수준을 위하여 다중 스레드 (Thread)를 위한 유사 상태인 분할 (Fork)과 합류 (Join)를 제공하고 있다. BPSS도 이 수준을 위해서 동일하게 분할 (Fork)과 합류 (Join) 요소를 갖고 있다. 완전 작업 수준은 기본 작업 수준과 중간 작업 수준들에 속한 요소들에게 속성 (Attribute)을 부여할 수 있도록 한다. 그 외의 구조적 작업 수준, 추가 구조적 작업 수준, 완전 구조적 작업 수준은 UMM이나 BPSS에서 제공하지 않는다. 구조적 작업 수준은 중간 작업 수준과는 서

로 겹치지 않고 공존할 수 있는 수준이다. 구조적 작업 수준은 모델을 읽고 유지보수 하기 쉽도록 도와주는 요소들을 포함하고 있다. 1960년대부터 GO-TO문으로 대표되는 스파게티 코드에 의한 소프트웨어 위기 (Software Crisis)를 해결하기 위하여 구조적 방법론을 사용하도록 권고되어 왔다[15]. 구조적 방법론은 모든 알고리즘을 오직 순차 (Sequencing), 조건 (Condition), 반복 (Iteration)만으로 서술한다. 그리고 각 조건문과 반복문 블록 (Block)들은 중첩될 수 있지만 교차될 수가 없다. 그러므로 그 블록들의 입구와 출구는 오직 하나씩이다. 그러나 UML의 상태들은 전이들에 의해 마치 스파게티 면발처럼 서로 얽힐 수 있어서 GO-TO문을 가진 프로그램 코드 같다. UML 1.5부터는 구조적 프로그램 형식의 모델링을 할 수 있는 구조적 단계를 위해 루프 (Loop)와 조건 (Conditional) 같은 개념들을 도입하고 있다.



〈그림 5〉 비구조적 선택과 구조적 선택

〈그림 5〉는 기존의 요소들만을 가지고 선택적 제어를 모델링할 때에 비하여 구조적 선택을 위한 요소를 제공할 때 가독성 (Readability)이 증가됨을 보여주고 있다. 현재의 UMM은 결정 (Decision) 노드를 가지고 선택적 제어를 모델링하기 때문에 〈그림 5a〉와 같이 블록들이 서로 교차되는 비구조적 선택이 가능하다. 이와 같은 비구조적 선택이 가능할 경우 작업의 수가 증가될수록 제어들이 서로 엉켜 가독성이 기하급수적으로 떨어지고 모델의 유지보수 비용도 증가되게 된다. 그러나 〈그림 5b〉와 같이 구조적 선택을 위한 요소를 제공할 경우 조건에 따른 작업의 흐름을 파악하기가 용이해진다. UML 2.0에서는 구조적 선택 제어를 위해서 〈그림 5b〉와 같이 조건 노드(Conditional Node)를 제공한다. 조건 노드는 조건에 따라 분기가 되는 여러 개의 절 (Clause)을 가지고 있다. 그리고 각 절은 검사부 (Test Part)와 본문 (Body)으로 이루어져 있다. 검사부가 만족되면 본문이 수행된다. 만일 여러 절의 본문 중 하나라도 실행이 되면 다른 절의 내용은 무시되는 데, 어느 절의 검사부를 먼저 검사할지는 미리 결정되어 있지 않다.

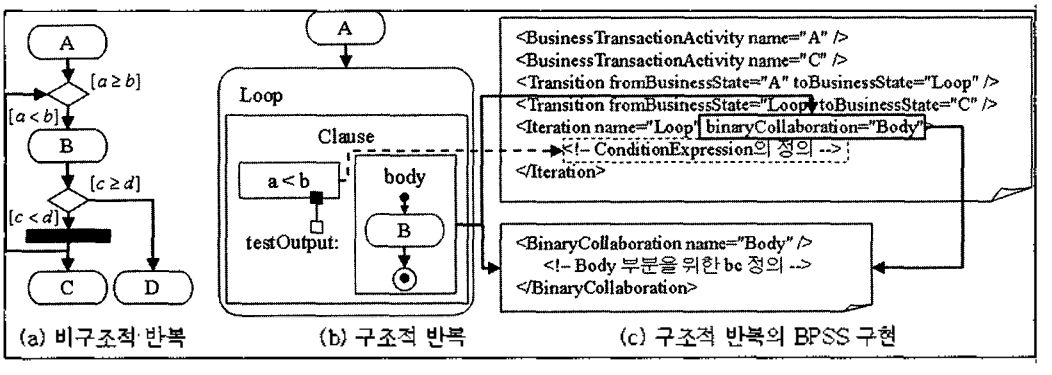
그러므로 구조적 선택 제어를 위해서는 UML의 조건 노드와 절에 해당하는 새로운 요소들을 BPSS에 도입하여야 한다. 먼저 UML의 조건 노드를 위해서는 선택 요소 (Selection Element)를 추가하였다. 선택 요소는 조건 노드와 유사하게 절에 해당하는 경우 요소 (Case Element)를 2 개 이상 가지고 있다. 경우 요소는 절의 선택부와 본문에 해당하는 조건식 (Conditional Expression, [018])과 BC 요소 (BinaryCollaboration Element, [021]~[023])를 이용한다. 본문을 정의하기 위하여 기존의 BC 요소를 사용하는 것은 여러 가지 장점이 있다. 먼저 기존 BC 모델들을 재사용할 수 있을 수 있다. 또한 선택적 제어가 포함되어 있는 BC 모델 내에 본문의 한 요소로써 BC 모델을 포함하여 계층적 모델링이 가능해 진다. 구조적 조건의 선택 요소 ([001]~[014])와 경우 요소 ([015]~[029])를 위한 BPSS의 XML 스키마를 정리하면 다음과 같다.

```
[001] 〈xsd:element name = "Selection"〉
[002]     〈xsd:complexType〉
[003]         〈xsd:sequence〉
[004]             〈xsd:choice minOccurs = "2"〉
```

```
[005]            〈xsd:element ref = "Case"/〉
[006]          〈/xsd:choice〉
[007]        〈/xsd:sequence〉
[008]        〈xsd:attributeGroup ref = "name"/〉
[009] 〈/xsd:complexType〉
[010] 〈xsd:unique name = "Selection-ID"〉
[011]        〈xsd:selector xpath = "."/〉
[012]        〈xsd:field xpath = "nameID"/〉
[013]   〈/xsd:unique〉
[014] 〈/xsd:element〉
[015] 〈xsd:element name = "Case"〉
[016]   〈xsd:complexType〉
[017]      〈xsd:sequence〉
[018]           〈xsd:element ref =
                "ConditionExpression"/〉
[019]      〈/xsd:sequence〉
[020]      〈xsd:attributeGroup ref = "name"/〉
[021]      〈xsd:attribute name =
           "binaryCollaboration"
[022]      type = "xsd:string" use = "required"/〉
[023]      〈xsd:attribute name =
           "binaryCollaborationIDREF"
           type = "GUIDREF"/〉
[024]   〈/xsd:complexType〉
[025]   〈xsd:unique name = "Case-ID"〉
[026]           〈xsd:selector xpath = "."/〉
[027]           〈xsd:field xpath = "nameID"/〉
[028]   〈/xsd:unique〉
[029] 〈/xsd:element〉
```

UMM에서는 반복에 어떠한 구조적 제한도 없기 때문에 〈그림 6a〉와 같이 대부분의 반복을 표현할 수 있다. 〈그림 6a〉와 같은 모델은 현재 구조적 방법으로는 표현이 불가능하다. 이러한 구조적 제한의 결여는 언어의 표현력을 증가시키는 장점이 있는 반면에 가독성과 유지보수 비용을 증가시킨다는 단점이 있다. 그러므로 특별히 비구조적인 반복이 필요하지 않다면 반복은 언제나 구조적인 형태를 갖추도록 제한을 가할 수 있는 방법이 필요하다. UML 2.0에서는 구조적 반복으로 제약할 수 있도록 〈그림 6b〉와 같은 루프 (Loop) 노드를 제공하고 있다. 루프 노드는 조건 노드와 유사하게 조건부와 본문으로 이루어져 있는 절을 가지고 있지만 루프는 오직 하나의 절만을 가진다는 면이 차별된다. 즉, 루프는 하나의 절 내의 조건부가 실패할 때까지 이 절의 본문을 반복해서 실행한다.

본 논문에서는 〈그림 6b〉와 같은 구조적 반복을 실현할 수 있도록 BPSS에 반복 요소 (Iteration Element)를 제안한다. 반복요소는 다음 XML 스키



〈그림 6〉 비구조적 반복과 구조적 반복

마와 같이 경우 요소와 유사하게 조건식 ([004])과 BC 요소 ([007]~[009])를 가지도록 설계하였다 〈그림 6c〉는 제안된 반복요소를 이용하여 〈그림 6b〉의 구조적 반복을 BPSS로 구현한 예이다. "

```
[001] <xsd:element name="Iteration">
[002]   <xsd:complexType>
[003]     <xsd:sequence>
[004]       <xsd:element ref=
              "ConditionExpression"/>
[005]     </xsd:sequence>
[006]     <xsd:attributeGroup ref="name"/>
[007]     <xsd:attribute name=
            "binaryCollaboration"
[008]     type="xsd:string" use="required"/>
[009]     <xsd:attribute name=
            "binaryCollaborationIDREF"
            type="GUIDREF"/>
[010]   </xsd:complexType>
[011]   <xsd:unique name="Iteration-ID">
[012]     <xsd:selector xpath="."/>
[013]     <xsd:field xpath="nameID"/>
[014]   </xsd:unique>
[015] </xsd:element>
```

UML의 완전 구조적 작업 수준은 조건 노드나 루프 노드에 조건식을 추가하기 위하여 데이터 출력 핀 (Data Output Pin) 개념을 도입하였다. BPSS에서는 절 요소에 조건식을 이미 추가하였으므로 구조적 작업 단계를 지원하는 동시에 완전 구조적 작업도 지원한다.

추가 구조적 작업 수준은 예외 상황 처리를 위한 요소 및 개념들을 제공한다. 추가 구조적 작업 수준은 ebBP가 기존의 BPSS에 비해 우월한 점 중에 하나이다. ebBP는 시간제약 외에도 다양한 예외사항을 미리 정의하고 있으며 거래 당사자들 간의 합의가 있을 경우에는 BT 작업 도중이라도 NOF (Notification of Failure)를 이용하여 작업을 종료시킬 수가 있다.

5. 인수 집합 (Parameter Set)

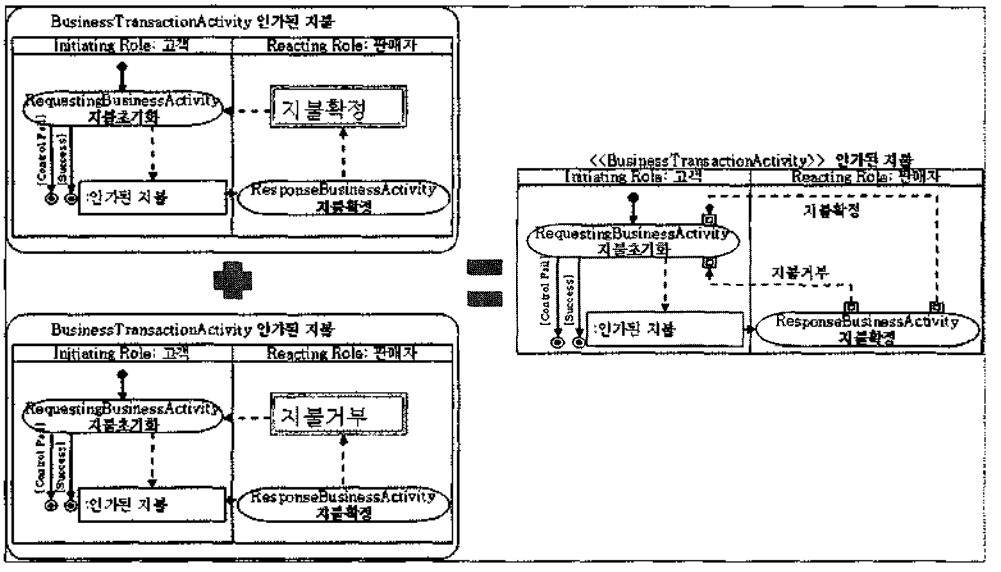
본 절에서 고려할 개념은 하나의 작업에 여러 종류의 입·출력이 가능할 때 모델을 단순화하기 위한 개념에 관한 것이다. UML 2.0에서는 작업의 입·출력이 핀 (Pin)이라는 것을 통해 이루어진다. 핀은 작업에 값들을 보내고 작업의 결과를 받는다. 일부 작업들은 다양한 종류의 입·출력 후보들을 허용하고 실제 작업 실행 중에는 입·출력의 집합에서 하나를 선택할 수 있도록 하고 있다. 이런 입·출력들의 후보 집합을 인수 집합 (Parameter Set)이라고 한다. 이 인수 집합을 이용하면 입·출력 종류별로 만들었던 여러 개의 작업도를 하나로 요약할 수 있다. 〈그림 7〉은 인수 집합을 이용하여 2 개의 BT가 하나의 BT로 통합되는 예를 보여준다. 기존의 BT는 ResponseBusinessActivity가 "지불 확정" 혹은 "지불 거부" 중 하나의 답변만을 표현할 수 있기 때문에 답변 부분만을 제외하면 모든 요소가 동일한 2개의 BT를 모델링해야 한다. 그러나 〈그림 7〉과 같이 인수 집합을 이용하여 2 개의 BT를 통합하면 성공적인 지불에 대해서는 지불 확정을, 실패의 경우에는 지불 확정 거부를 위한 봉투 (Envelope)를 보내도록 모델링을 할 수 있어 시스템의 가독성은 늘고 모델 수는 감소된다.

BPSS의 BT에 인수 집합의 개념을 도입하기 위해서는 ResponseBusinessActivity가 RequestingBusinessActivity에게 보내는 봉투 (Envelope)의 개수를 0 개에서 무한대까지 가능하도록 허용해야 한다. 다행히 현재의 ResponseBusinessActivity도 복수 개의 봉투를 허용하고 있다. 그러나 이 복수 개의 봉투를 어떻게 해

석해야 하는지에 대한 합의는 이루어져 있지 않은 실정이다. 그러므로 이 복수 개의 봉투들은 모델링 했을 때에 한 번에 여러 개의 봉투를 보낸다는 의미인지 아니면 여러 개의 봉투 중의 하나를 보낸다는 의미인지가 불명확했다. 본 논문에서는 이 여러 개의 봉투를 인수 집합의 개념으로 해석하여 전체 모델을 간략하게 만들 수 있도록 제안한다. 대신 만일 여러 개의 봉투를 보내야 하는 상황이라면 그 봉투들을 포함하는 새로운 봉투를 만들어 보내는 것으로 구현가능하다.

6. 결 론

ebXML에서는 전자거래를 위한 비즈니스 프로세스를 모델링하고, 구현하기 위한 언어로 UMM과 BPSS를 권고하고 있다. 이 두 모델링 언어는 UML 1.4를 기반으로 하며, BPSS는 XML을 이용하여 정의되어 있다. ebBP의 경우 BPMN을 기반으로 하고 있기는 하지만 UMM으로 모델링된 비즈니스 프로세스를 표현할 수 있어야 하므로 UML 2.0의 표현력을 포함해야 한다. 최근의 UML은 비즈니스 프로세스의 모델링을 강화할 수 있는 방향으로 향상되고 있다. 그러므로 UML의 새로운 요소를 UMM에 도입할 경우 표현력과 유지보수의 용이성을 증대시킬 수 있지만 BPSS로 구현이 불가능할 수 있다. 그러므로 본 논문에서는 UMM을 향상시킬 수 있는 UML 2.0의 작업도 개념들을 발굴하고 이 개념들을 BPSS로 구현하는 방안을 제시하였다. 본 논문에서 고려한 개념들 외에도 UML 2.0은 비즈니스 프로세스를 정의하는 데 유용한 개념들을 새로 도입하였으므로 ebXML의 발전을 위해서는 이러한 개념들을 고찰, 분석하는 연구가 필요하다.



〈그림 7〉 인수 집합을 이용하여 요약된 BT

참 고 문 헌

[1] 강현구, 천두완, 김수동, "UML 2.0 기반 객체 지향 모델링 프로세스 및 지침," 한국정보과학회 가을 학술발표논문집, 제 31권, 21호, 307-309, 2004.

[2] 고현민, 손명근, 오윤주, 배두환, "The Analysis on the Possibility of Business Process Modeling using UML Activity Diagram," 한국정보과학회 학술발표논문집, pp. 112-114, 2003.

[3] 산업자원부, e-Biz 표준화 백서 2004, 산업자원부, 2004.

[4] 임태수, "XMI를 활용한 비즈니스 프로세스 모델 호환 방법론," 한국전자거래학회, 제 11권, 제 3호, 73-88, 2006.

[5] 한국전자거래진흥원, 2006년 상반기 UN/CEFACT 표준화 작업동향 보고서, 한국전자거래진흥원, 2006.

[6] Altova, XMLSpy 2005 User&Reference Manual, Vervante, 2006.

[7] Dumas M., Ter Hofstede A.H.M., "UML Activity Diagrams as a Workflow Specification Language," Lecture Notes in Computer Science, Vol. 2185, pp.76-90, 2001.

[8] Gardner T., "UML Modeling of Automated Business Processes with a Mapping to BPEL4WS," in Proceedings of 17th European Conference on Object-oriented Programming, 2003.

[9] ISO, Open-edi Reference Model: ISO/IEC JTC 1/SC30 ISO Standard 14662 ISO, 1995.

[10] Kim J.-H., Huemer C., "Analysis, Transformation and Improvements of ebXML Choreographies Based on Workflow Patterns," in Proceedings of the International Conference on Cooperative Information Systems (CoopIS 04), pp. 66-84, 2004.

[11] Kim J.-H., Huemer C., "From an ebXML BPSS choreography to a BPEL-Based Implementation," ACM SIGecom Exchange, Vol. 5, No. 2, pp. 1-11, 2004.

[12] Murata T., Yamauchi H., "A Petri Net with Negative Tokens and its Application to Automated Reasoning," in Proceedings of the 33rd Midwest Symposium on CAS, pp.762-765, August 1990.

[13] OASIS, ebXML Business Process Specification Schema Technical Specification v2.0.3, OASIS, 2006.

[14] OMG, UML 2.0 Superstructure Specification: OMG Adopted Specification ptc/03-08-02, OMG, 2003.

[15] Pressman, R. S. Software Engineering: A Practitioner's Approach, McGrawHill, 2004.

[16] UN/CEFACT TMG, ebXML Business Process Specification version 1.10, 2003.

[17] Van der Aalst W.M.P, Ter Hofstede A.H.M., Kiepuszewski B., Barros A.P., "Workflow Patterns," Distributed and Parallel Databases, Vol. 14, pp. 5-51, 2003.

[18] Wohed P., Van der Aalst W.M.P., Dumas M., Ter Hofstede A.H.M., Russell N., "Pattern-based Analysis of UML Activity Diagrams," BETA Working Paper Series, WP 129, Eindhoven University of Technology, Eindhoven, 2004.

저 자 소 개

김자희	(E-mail: jahee@snut.ac.kr)
1995.	한국과학기술원 전산학과
1997.	한국과학기술원 전산학과(석사)
2003.	한국과학기술원 산업공학과(박사)
2004.	University of Vienna Visiting Researcher
2005~현재	서울산업대학교 IT정책전문대학원 전임강사
관심분야	전자상거래, 비즈니스 프로세스, 의료정보시스템, 시맨틱 웹서비스