워크플로우 프로세스 정의 교환을 위한 XPDL 메타모델의 모델링

(Modeling of XPDL Meta-Model for Workflow Process Definition Interchange)

김 진 성 † 유 춘 식 †† 김 용 성 †††
(Jin Sung Kim) (Chun Sik Yoo) (Yong Sung Kim)

요 약 XPDL(XML Process Definition Language)은 프로세스 정의 교환을 위하여 WfMC에서 제안한 XML 기반 언어이다. 본 논문에서는 워크플로우 프로세스의 정의와 교환을 위하여 XPDL 문서의 구조(Schema)를 UML 다이어그램으로 모델링하여 기업들 간의 상호연동과 협업을 위한 업무흐름 파악을 용이하게 하는 모델을 제안한다. 이를 위해 XPDL 문서를 UML의 클래스 다이어그램과 액티비티 다이어그램으로 변환하는 사상 규칙를 정의하여, XPDL 프로세스 메타모델(Meta-Model)를 구성하는 각 엘리먼트는 클래스 다이어그램으로 모델링하고, 프로세스 액티비티(Process Activity)는 액티비티 다이어그램으로 모델링하는 기법을 제안한다. 또한 제안된 기법을 "신용 카드 상태 체크 시스템"의 워크플로우에 대한 XPDL 문서을 적용하여 제안된 기법의 유효성을 검증한다.

키워드 : 워크플로우, XPDL, 프로세스 메타모델, XML Schema, 객체 모델링, 액티비티 모델링

Abstract XPDL is a XML-based language for process definition exchange that is proposed by WfMC. This paper propose a model which model XPDL document structure (Schema) using UML in order to define and to exchange workflow process, and make business flow understanding ease for inter-business cooperation. So, we define mapping rules in which map XPDL documents into UML class diagram and UML activity diagram. By these mapping rules, elements composing XPDL process meta-model are mapped into UML class diagram, and process activities are mapped into UML activity diagram. Also, we apply proposed mapping technique to model a workflow of "Credit card state check system."

Key words : Workflow, XPDL, Process Meta-Model, Object Modeling, Activity Modeling

1. 서 론

오늘날 기업에서 사용되는 정보시스템은 업무의 형태나 처리 방법이 기업 내부는 물론 다른 기업들과 연계되면서 표준화, 상호연동, 분산처리 등의 기술요소가 중요한 이슈로 부상하고 있다. 즉 기존의 단일 기업에 대한 독자적인 기업 활동이 아니라 동종 또는 이종 기업 간의 상호 연동과 협업을 통해서 기업 의 이윤 창출을 모색하고 있다[1]. 특히 이와 같은 노력들은 전자상거래에서 고객과 공급업체 그리고 비즈니스 파트너들 간의 협업과 정보 공유를 수행하는 일련의 비즈니스 프로세스(Business Process)에서 흔히 찾아볼 수 있다. 여기서 비즈니스 프로세스란 기업 간 전자상거래를 위한 업무 처리 절차를 말하며, 전자상거래의 정보흐름을 '제어한다는 의미에서 전자상거래 구현의 핵심적인 역할을 수행한다[2]. 따라서 기업 간의 상이한 문서 양식과 프로세스에 대한 표준화를 통해 전자상거래의 효율성과 생산성을 극대화할 수 있다.

이러한 비즈니스 프로세스의 표준화를 위한 대표적인 기술이 워크플로우(Workflow)[3]이다. 워크플로우 기술이란 특정 기업이나 조직체 내부 또는 조직체 사이에서 발생하는 일련의 업무의 흐름을 정의하고, 업무가 정해진 시간 안에 관련 정보와 함께 자동적으로 수행되도록

· 이 논문은 2004년도 한국학술진흥재단의 지원에 의해서 연구되었음 (KRF-2004-042-D00168)
† 학생회원 : 전북대학교 컴퓨터통계정보학과
kpjiju@chonbuk.ac.kr
†† 종신회원 : 전북대학교 전산통계학과
csyoo@chonbuk.ac.kr
††† 종신회원 : 전북대학교 전자정보공학부 교수
yskim@chonbuk.ac.kr
논문접수 : 2005년 3월 10일
심사완료 : 2005년 4월 18일

제어하는 분산 컴퓨팅 기술을 의미한다[4]. 이러한 워크플로우 기술의 도입과 적용은 기업의 경쟁력 자체를 증가시키는 중요한 요소가 될 수 있다. 이러한 이유로 최근 대부분의 기업들은 기업 내의 모든 업무에 대해 일련의 업무흐름 절차를 표준화하여 관련된 정보, 지식, 서류 등을 정확하게 접수 및 분류하고, 이를 다시 전달하는 형태로 업무를 수행하고 있다. 현재 워크플로우에 대한 표준은 워크플로우 관련 단체인 WfMC(Workflow Management Coalition)에서 발표한 표준이 통용되고 있다. 또한 이러한 워크플로우에 관련된 전자상거래, 공급망 관리, 전자물류, 전자조달, 웹 서비스 분야에서 메타 언어인 XML을 기반으로 호환성과 연동성이 뛰어난 XPDL의 이용 범위가 점차 확대되고 있다.

따라서 본 논문에서는 XPDL의 프로세스 메타모델을 UML(Unified Modeling Language)[5,6] 다이어그램으로 모델링하여 기업 간의 상호연동과 협업을 위한 업무흐름 파악을 용이하게 하는 모델를 제안한다. 이를 위해 XPDL 문서의 스키마를 UML의 클래스 다이어그램과 액티비티 다이어그램으로 변환하는 사상 규칙을 제안하고, 이를 "카드 상태체크 시스템"에 적용한다. 즉 "카드 상태체크 시스템"의 워크플로우에 대한 XPDL 스키마를 설계한 후, 각 엔티티들은 UML의 클래스 다이어그램으로 모델링하고, 전체적인 작업흐름은 액티비티 다이어그램으로 모델링하여 제안된 기법의 유효성을 검증한다.

2. 관련 연구

이 장에서는 XML 스키마와 XPDL 문서 모델링에 관련된 연구들을 비교·분석한다. 먼저 XML 스키마를 UML 클래스 다이어그램으로 모델링하는 연구들은 [5-7]이 있다. [5]는 XML 스키마의 주요 구조를 UML의 클래스 다이어그램으로 모델링하는 방법을 제안하였다. 본 논문에서는 [5]의 표현법을 기초로 클래스들 사이의 연관성을 추가하여 XML 스키마를 UML 클래스 다이어그램으로 변환하기 위한 규칙을 정립하였다. 그리고 [6]과 [7]은 XML 스키마의 주요 객체에 대하여 모델링 과정을 설명하고 있다. 특히 스테레오타입, 반복횟수, 클래스의 상속관계 등의 세부적인 모델링 과정을 제시하고 있다.

그리고 XPDL 문서 모델링에 대한 주요 연구들을 살펴보면, 워크플로우 개념을 이용한 생산 시스템에서 비즈니스 프로세스를 UML 액티비티 다이어그램으로 표현한 연구[8], 분산된 협력적 워크플로우를 UML 다이어그램으로 설계한 연구[9] 그리고 XPDL에서 제공되는 메타모델과 관련 문법을 정립하고 XPDL 시스템을 액티비티 다이어그램으로 모델링한 연구[10] 등이 있다.

[8]은 워크플로우 개념이 적용된 생산 시스템을 구성하고 있는 각 엔티티를 UML 클래스 다이어그램으로 표현하고, 세부적인 프로세스의 행위들은 UML 액티비티 다이어그램으로 모델링하였다. 이 논문에서는 특정 시스템의 업무흐름을 액티비티 다이어그램 위주로 모델링하였기 때문에 구체적인 사상 기법에 대한 언급이 부족하다. [9]는 시스템에 적용된 XPDL 프로세스 메타모델에 대한 각 엔티티들을 사상 테이블로 상세하게 기술하였지만 사상 테이블을 적용하여 다이어그램으로 모델링하는 부분이 없어 정형화할 수 없는 단점이 있다. [10]은 XPDL에서 제공된 프로세스 메타모델과 관련 문법을 정립하고 각 엔티티들의 업무흐름에 대한 행위를 UML 액티비티 다이어그램으로 사상하는 사상 테이블을 작성한 후, 액티비티를 모델링하였다. 이 논문에서는 사상 대상을 액티비트에 대해서만 제한하고 있기 때문에 표현 범위가 너무 한정된다는 단점이 있다.

따라서 본 논문에서는 XPDL 프로세스 메타모델에 대한 엔티티와 프로세스의 행위를 모델링하기 위한 사상규칙과 사상 테이블을 정의하고, 이를 기반으로 UML의 클래스 다이어그램과 액티비티 다이어그램을 이용하여 XPDL 프로세스 메타모델에 대한 객체 모델링과 액티비티 모델링을 수행하기 위한 방안을 제안한다. 본 논문에서 제안된 모델링 기법을 사용하면 XPDL 프로세스 메타모델에 대한 정형화된 표현 방법을 제공할 수 있으며 워크플로우 프로세스 모델링의 전과정에 걸쳐 적용할 수 있다.

3. UML, XPDL, XML 스키마 모델링

본 장에서는 UML과 XPDL의 구성요소 그리고 XML 스키마를 모델링하는 기법에 대하여 기술한다.

3.1 UML(Unified Modeling Language)[11]

UML의 다이어그램 중에서 본 논문에서 이용하는 클래스 다이어그램과 액티비티 다이어그램에 대해서 기술한다.

3.1.1 UML 클래스 다이어그램

UML 클래스 다이어그램은 클래스와 그들 간의 관계로 구성된다. 다음 각 절에서 UML 클래스 다이어그램을 구성하는 각 구성요소에 대하여 기술한다.

(1) 클래스

클래스는 클래스 이름, 속성(Attribute) 그리고 오퍼레이션(Operation) 부분으로 구성된다. 속성은 클래스의 특성을 표현하는 변수들이며, 오퍼레이션은 클래스의 행위들을 표현하는 함수(메소드)들이다.

(2) 집단화(Aggregation) 관계

집단화 관계는 상위 클래스와 하위 클래스의 관계가 'part-of' 관계임을 나타내며 '◇'로 표기한다. 2 개 이상의 집단화 관계에서 선택적으로 하위 클래스가 발생하는 경우 'xor' 관계를 갖는다. 'xor' 관계는 여러 개의

집단화 관계를 점선으로 연결하고 '{xor}'이라는 제한조건을 준다.

(3) 일반화(Generalization)/특수화(Specialization)

일반화는 공통된 속성이나 연산을 가진 클래스들에서 공통 요소들을 추출하여 상위 클래스를 만들어내는 것을 의미하며, 이와 반대로 상위 클래스에 새로운 요소를 추가하여 하위 클래스를 만들어내는 것을 특수화라고 한다. 클래스를 실선으로 연결하고 상위 클래스 쪽에 빈 삼각형을 표시한다.

(4) 의존(Dependency) 관계

한 클래스의 변화가 다른 클래스에 영향을 주는 관계로, 점선으로 표시하며 방향성을 나타낼 수 있다.

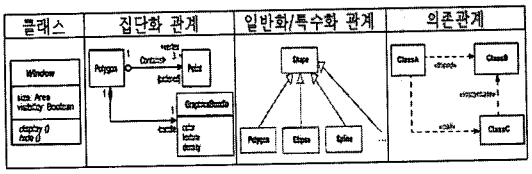

그림 1 UML 클래스 다이어그램의 구성요소

3.1.2 UML의 확장 메커니즘

UML은 여러 가지 객체지향 기법을 결합한 형태로 다양한 확장 메커니즘을 제공한다. 제한조건, 스테레오타입 등이 이에 해당된다.

(1) 제한조건(Constraints)

제한조건은 텍스트로 표현된 의미적 조건이나 제약사항으로, 모델 요소들 사의의 의미적 관계이다. 제한조건은 항상 참이 되어야 하며, 표기는 '{ }' 안에 조건을 나타내는 텍스트 스트링을 기술한다.

(2) 스테레오타입(Stereotypes)

스테레오타입은 기존의 UML 요소를 기반으로 UML 어휘를 확장하여 새로운 메타 클래스의 인스턴스를 만들기 위해 고안되었다. 즉, 스테레오타입은 기존의 메타모델 요소와 같은 형태를 가지나 다른 의도로 사용되는 메타모델 요소의 하위 클래스이다. 스테레오타입은 '« »' 안에 스테레오타입의 이름을 키워드 스트링으로 기술하고 이를 모델 요소의 이름 앞이나 위에 표기한다.

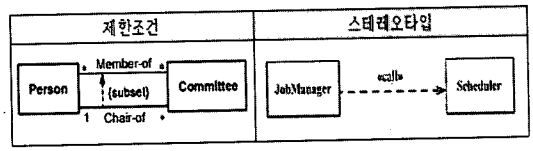

그림 2 UML의 확장 메커니즘

3.1.3 UML 액티비티 다이어그램

UML 액티비티 다이어그램은 액티비티의 제어 흐름을 시각화여 보여준다. 다음 각 절에서 UML 액티비티 다이어그램을 구성하는 각 구성요소에 대하여 자세히 기술한다.

(1) 액티비티(Activity)

액티비티란 시스템에서 수행되는 작업 단위를 의미하며, 액티비티의 이름을 포함하는 둥근 사각형으로 표시된다.

(2) 전이(Transition)

한 동작이나 활동 상태에서 다른 동작이나 활동상태로 가는 경로를 나타내며, '→'로 표시된다.

(3) 분기(Branching)

하나의 전이가 부울식에 의해서 두 개 이상의 액티비티로 전이되는 것을 의미하며, '◇'를 이용하여 표시한다.

(4) 분할(Split)과 결합(Join)

동기화 막대를 이용하여 동시 제어 흐름을 여러 개로 나누거나 하나로 합하는 전이의 표현을 의미한다.

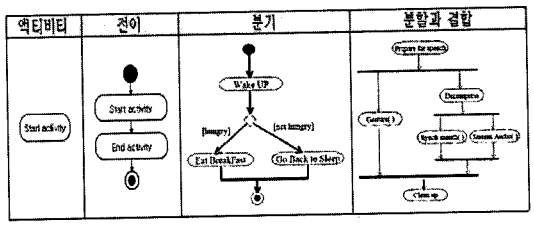

그림 3 UML 액티비티 다이어그램의 구성요소

3.2 XPDL[12]

XPDL은 비즈니스 프로세스에 대한 정의 및 교환을 목적으로 WfMC에서 제안한 XML을 기반으로 한 언어이다. 이 절에서는 XPDL의 개요와 XPDL의 메타모델을 구성하는 각 엔티티의 기능과 구성요소에 대하여 알아본다.

3.2.1 WfMC 워크플로우 참조 모델과 XPDL

WfMC에서 발표한 워크플로우 참조모델[13]은 워크플로우를 수행하는 엔진과 내부 동작을 지원하는 5 가지 API를 제공하고 있다. 이와 같은 워크플로우 참조모

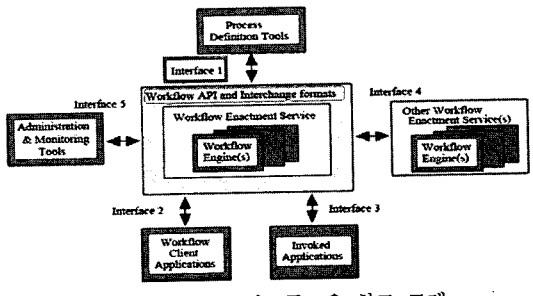

그림 4 WfMC 워크플로우 참조 모델

델은 가장 일반적인 워크플로우의 구조를 표현하고 있으며, 대다수 워크플로우 관련 업체들은 워크플로우를 설계할 때 이를 기반으로 하거나 참조하고 있다. 특히 프로세스를 정의하고 정의된 프로세스를 상호 교환하기 위한 "Interface 1"에 대하여 XML 기반의 프로세스 정의 언어인 XPDL를 개발하여 사용하고 있다.

3.2.2 XPDL 프로세스 메타모델

XPDL의 프로세스 메타모델은 비즈니스 프로세스를 정의하고 교환하기 위해, 다음 그림과 같이 엔티티를 정의하고 선언하는 Workflow Process Definition 엔티티와 하위 5 개의 엔티티로 구성된다.

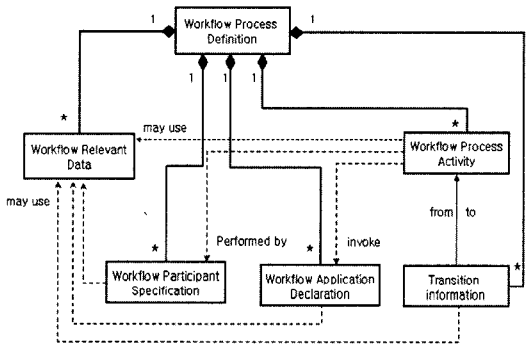

그림 5 XPDL Process Meta-Model

(1) 워크플로우 프로세스 정의(Workflow Process Definition) 엔티티

워크플로우 프로세스 정의 엔티티는 하위 5 개의 엔티티인 Applications, Participants, DataFields, Activities, Transitions을 정의하고 선언하는 역할을 수행한다. 다음은 XPDL에서 제공하는 워크플로우 프로세스 정의 엔티티를 구성하는 구성요소이다.

표 1 워크플로우 프로세스 정의 엔티티의 구성요소

구성요소		내 용
엘리먼트	Activities	프로세스를 수행하는 액티비티 정의
	Participants	액티비티를 수행하는 시스템 또는 사람 정의
	Applications	프로세스 수행에 필요한 어플리케이션 정의
	DataFields	프로세스 수행에 필요한 데이타 정의
	Transitions	액티비티의 실행 제어 조건 정의
속성	Id	워크플로우 프로세스간을 구분하는 기능
	AccessLevel	외부 시스템 또는 어플리케이션에 의해서 워크플로우 프로세스 정의를 호출할 수 있는 접근 레벨을 기술(Public 또는 Private)

(2) 워크플로우 프로세스 액티비티(Workflow Process Activity) 엔티티

액티비티는 프로세스의 기본단위, 즉 비즈니스를 수행하는 작업 단위를 의미하며, 액티비티 수행에 있어서 IT(Information Technology) 어플리케이션을 호출하고 관련 데이타를 사용한다. 또한 파티시펀트에서 정의된 시스템 또는 사람을 실행시키는 엔티티이다. 다음은 액티비티 엔티티의 구성요소이다.

표 2 액티비티 엔티티의 구성요소

구성요소		내 용
엘리먼트	Description	액티비티의 기능을 기술
	Route	더미 액티비티 표현
	Implementation	정규(regular) 액티비티 표현
	BlockActivity	액티비티 집합(Activity set)을 실행하는 액티비티 표현
	Performer	액티비티의 참여자를 기술한다. (시스템 또는 사람
	TransitionRestrictions	액티비티의 분열 또는 결합의 제약조건을 기술한다.
	Activities	프로세스 영역에 관련된 다수의 액티비티를 포함하고 실행한다.
속성	Id	액티비티간의 구분

액티비티는 다음과 같이 3가지의 구조를 가지고 있다.

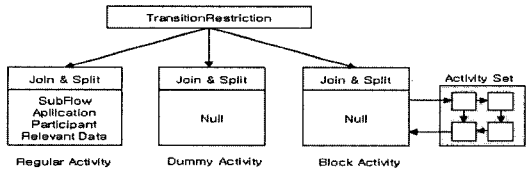

그림 6 액티비티의 구조와 제약 조건

액티비티들은 전이 제약조건(Transition Restriction)에 의해서 분할 또는 결합될 수 있다. "Implemetation"에서 표현되는 정규 액티비티는 몸체 부분에 다른 워크플로우 프로세스 정의, 어플리케이션 및 액티비티 참여자 그리고 관련 자료를 포함하고 있는 정규 액티비티이다. 한편 "Route"에서 표현되는 더미 액티비티는 몸체 부분이 비여있는 상태이다. 즉 더미 액티비티는 관련 데이타나 어플리케이션 등이 필요없는 액티비티로, 단지 액티비티의 시작과 끝을 나타낸다. 블록 액티비티도 몸체 부분이 비여 있으며, 다수의 액티비티와 트랜지션을 포함하고 있는 액티비티 집합을 나타낸다.

(3) 워크플로우 파티시펀트 기술(Workflow Paticipant Specification) 엔티티

파티시펀트 기술 엔티티는 다양한 액티비티의 수행자로 참여할 수 있는 자원을 기술하는 엔티티이다. 여기서 자원은 시스템 또는 사람이 될 수 있다. 다음은 파티시펀트 엔티티의 구성요소이다.

표 3 파티시펀트 기술 엔티티의 구성요소

구성요소		내 용
엘리먼트	Description	파티시펀트의 기능을 기술
	ParticipantType	시스템 또는 사람
	Participants	다수의 파티시펀트를 포함하고 실행한다.
속성	Id	파티시펀트간의 구분

(4) 워크플로우 어플리케이션 선언(Workflow Application Declaration) 엔티티

"Application"은 액티비티 수행에 필요한 IT 어플리케이션을 선언하는 엔티티이다. 액티비티는 어플리케이션의 사용을 위해 Id 이름과 파라미터 정보를 전달하여 어플리케이션을 호출할 수 있다. 다음은 어플리케이션 선언 엔티티의 구성요소이다.

표 4 어플리케이션 선언 엔티티의 구성요소

구성요소		내 용
엘리먼트	Description	애플리케이션의 기능을 기술
	FormalParameters	액티비티에서 파라미터와 함께 어플리케이션을 호출할 때 수신받는 파라미터를 지정
	Applications	다수의 애플리케이션을 포함하고 실행
속성	Id	애플리케이션간을 구분하는 기능

(5) 워크플로우 릴리번트 데이타(Workflow Relevant Data) 엔티티

릴리번트 데이타 엔티티는 프로세스 수행 중에 각 프로세스 인스턴스에서 사용되거나 생성될 자료들의 타입, 초기값, 크기 등을 정의하는 엔티티이다. XPDL에서는 <DataField> 엘리먼트로 표현된다. 다음은 데이타필드 엔티티의 구성요소이다.

표 5 데이타 필드 엔티티의 구성요소

구성요소		내 용
엘리먼트	DataType	데이타의 유형(숫자형,문자형) 기술
	InitialValue	데이타의 초기값 지정
	Length	데이타의 크기 지정
	DataFields	다수의 데이타필드를 포함하고 실행
속성	Id	데이타필드간 구분

(6) 트랜지션 정보(Transition Information) 엔티티

트랜지션 정보 엔티티는 프로세스를 수행하는 액티비티들의 실행 제어조건을 기술하는 엔티티이다. "From"은 시작되는 액티비티를 의미하고, "To"는 종료되는 액티비티를 의미한다. 다음은 트랜지션 정보 엔티티의 구성요소이다.

표 6 트랜지션 정보 엔티티 구성요소

구성요소		내 용
엘리먼트	Condition	트랜지션을 수행하기 위한 조건을 기술
	From	시작 액티비티를 지정
	To	종료 액티비티를 지정
	Transition	다수의 트랜지션을 포함하고 실행
속성	Id	트랜지션간의 구분

3.2.3 XPDL 팩키지 메타모델

XPDL 팩키지 메타모델은 동일 영역에서 워크플로우를 수행하는 다수의 워크플로우 프로세스들을 하나의 워크플로우 프로세스 이름으로 묶여서 표현하는 모델이다. 따라서 아래 그림과 같이 워크플로우 프로세스 엔티티에는 다수의 워크플로우 프로세스가 포함되어 있으며, 이러한 프로세스가 이용하는 어플리케이션, 릴리번트 데이타, 파티시펀트는 각 프로세스 안에 정의된다. 또한 가장 상위에 정의된 워크플로우 팩키지에서도 어플리케이션, 렐러번트 데이타, 파티시펀트가 정의될 수 있는데, 이러한 엔티티들은 워크플로우 프로세스 안에 포함된 다수의 프로세스들이 이용하게 된다.

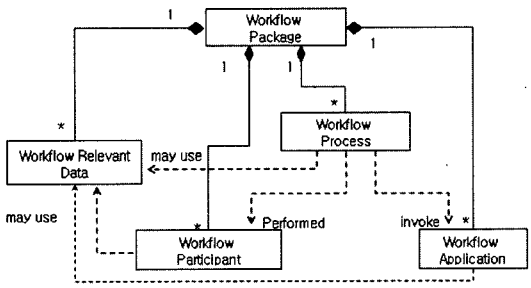

그림 7 XPDL Package Meta-Model

3.3 XML Schema[14] 모델링

XML Schema는 기존의 DTD에서 제공되지 못하는 기능을 확장·보완한 XML 문서구조 표현형식이다. 이 절에서는 XML Schema를 구성하는 엘리먼트의 역할과 특징을 살펴보고, XPDL 스키마 모델링에 적용하기 위하여 XML Schema 모델링에 대하여 알아본다.

3.3.1 XML Schema 구성 엘리먼트

XML Schema는 <Element>, <ComplexType>, <SimpleType>, <Group>, <AttributeGroup> 엘리먼트로 구성되며, 다음 표 7은 각 엘리먼트에 대한 역할과 특징을 나타낸다.

3.3.2 XML Schema 모델링

XML Schema의 <Element>, <ComplexType> 그리고 <SimpleType>을 UML 클래스 다이어그램으로

표 7 XML 스키마 구성 엘리먼트에 대한 역할과 특징

XML 시키마 엘리먼트	역할과 특징
<Element>	<Element>는 XML Schema의 최상위와 <ComplexType> 그리고 <Group> 엘리먼트에서 사용될 수 있다. 단, <SimpleType>안에서는 사용이 불가능하다.
<ComplexType>	<ComplexType>은 최상위 <Element>에 대한 타입과 속성을 정의하며, 하위 <Element>들을 정의하는 엘리먼트이다.
<SimpleType>	<SimpleType>은 하위 <Element>에 대한 단순형(SimpleType)을 선언하는 엘리먼트로, <SimpleType> 안에서는 엘리먼트나 속성을 사용할 수 없다.
<Group>	여러 개의 하위 <Element>들을 하나의 이름으로 그룹화하여 표현하는 엘리먼트로, 이것을 모델 그룹(Model Group)이라고 한다.
<Attribute Group>	여러 개의 속성들을 하나의 이름으로 그룹화하여 표현하는 엘리먼트로, 이것을 속성 그룹(Attribute Group)이라고 한다.

모델링하는 기법[5-7]에 대하여 알아본다.

(1) <Element> 엘리먼트

<Element>를 클래스 다이어그램으로 모델링하기 위하여, 클래스 이름 부분에 <<element>> 스테레오타입과 엘리먼트 이름을 표현하며, 엘리먼트가 참조하는 각 요소들에 대해서 <<has type>> 스테레오타입과 의존관계를 이용하여 참조 관계를 표현한다. 다음은 <Element>에 대한 예와 모델링 결과이다.

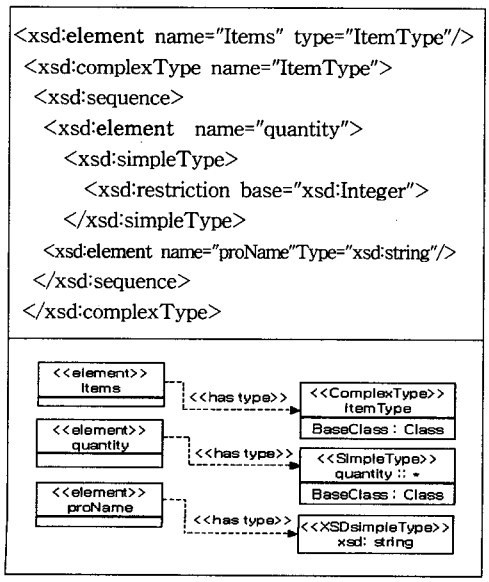

그림 8 <Element> 예와 모델링 결과

여기에서 최상위 엘리먼트인 "Items"는 다음에 위치한 <ComplexType> 안에 있는 요소들에 의해서 타입이 결정되며, <ComplexType> 요소 안에 있는 엘리먼트들도 다음에 위치한 <SimpleType>에 의해서 타입이 결정된다. 또한 "proName" 엘리먼트는 직접 단순형(string)을 선언한 경우이다.

(2) <ComplexType> 엘리먼트

<ComplexType>은 엘리먼트에 대한 타입과 속성 및 하위 엘리먼트 등을 정의하는 요소이다. 이를 UML 클래스 다이어그램으로 모델링하기 위하여, 클래스 이름 부분에 <<ComplexType>> 스테레오타입과 ComplexType의 이름을 표현하고, 클래스 속성 부분에는 ComplexType이 포함하고 있는 하위 엘리먼트들을 표현한다. 다음 그림은 <ComplexType>에 대한 예와 모델링 결과이다.

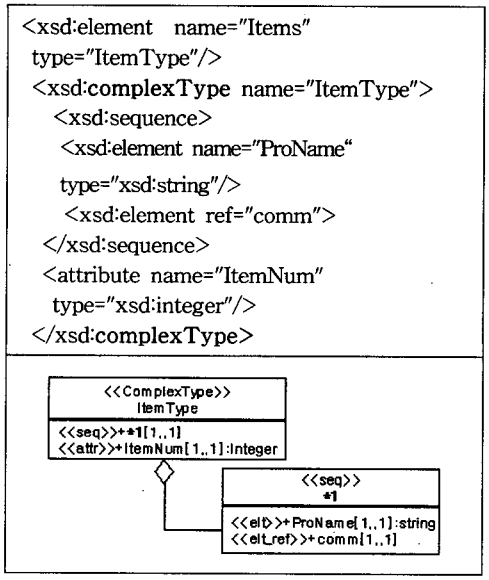

그림 9 <ComplexType> 예와 모델링 결과

<sequence> 엘리먼트는 <<seq>> 스테레오타입과 식별자로 표현하며, 식별자에 대한 표현 형식은 다음 절에서 자세히 기술한다. 또한 <attribute> 엘리먼트는 <<attr>> 스테레오타입을 사용한다. 한편 <sequence> 태그를 구성하는 두 개의 엘리먼트 중, 첫 번째 엘리먼트는 name과 단순형 타입을 직접 표현하는 형태로, <<elt>> 스테레오타입과 엘리먼트 이름, 발생횟수, data Type을 순서대로 표현하며, 외부 엘리먼트를 참조하는 두 번째 경우에는 <<elt_ref>> 스테레오 타입을 사용한다.

(3) <SimpleType> 엘리먼트

<SimpleType>은 엘리먼트에 대한 단순형을 선언하는 요소이다. 이를 UML 클래스 다이어그램으로 모델링하기 위하여, 클래스 이름 부분에 <<SimpleType>> 스테레오타입과 SimpleType의 이름을 표현하고, 클래스 속성 부분에는 SimpleType안에 단순형을 포함하고 있는 하위 엘리먼트들을 표현한다. 다음은 <SimpleType>에 대한 예와 모델링 결과이다.

```
<xsd:simpleType name="KState">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="JE"/>
    <xsd:enumeration value="SE"/>
    <xsd:enumeration value="GW"/>
  </xsd:restriction>
</xsd:simpleType>
```

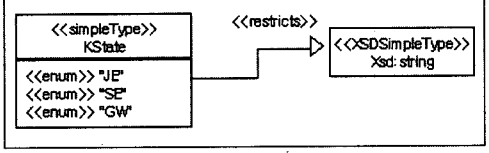

그림 10 <SimpleType> 예와 모델링 결과

KState의 이름을 가진 simpleType은 단순형인 string의 기본 타입을 가지며, 값은 <enumeration> 엘리먼트에서 지정한 "JE", "SE", "GW" 값으로 제한(restriction)하고 있다. 단순형에 대한 표현은 <<XSDSimpleType>> 스테레오타입과 단순형을 직접 기술하며, 열거형의 값은 단순형인 string의 클래스에서 상속되기 때문에 상속관계와 <<restricts>> 스테레오타입으로 표현한다.

(4) 식별자

식별자는 complexType이나 simpleType에 이름이 없는 경우 또는 <sequence>태그처럼 구분이 모호한 경우에 사용한다. 이를 UML 클래스 다이어그램으로 모델링하는 방법은 특정 엘리먼트안에 포함된 ComplexType의 이름이 없는 경우에는 "Element 이름 :: *"로 표현한다. 만약 ComplexType에 포함된 <sequence> 태그와 같은 경우에는 "* 식별번호"의 형식으로 표현하여 다른 이름이 없는 태그들과 구분한다. 다음은 식별자의 예와 모델링 결과이다.

```
<xsd:element name="price">
  <xsd:complexType>
    <xsd:attributename="currency"
     type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

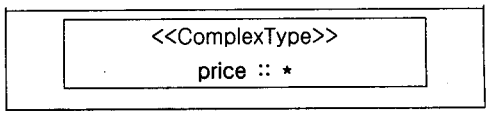

그림 11 식별자의 예와 모델링 결과

(5) 반복횟수

XML Schema에서 사용되는 발생횟수는 다음과 같다.

표 8 XML Schema 발생 횟수

발생횟수	규칙	의미
minOccurs=0, maxOccurs=생략	[0..1]	0에서 1회
minOccurs=0, maxOccurs="unbounded"	[0..*]	0에서 무한대
minOccurs=0, maxOccurs=생략 또는 생략	[1..1]	0회(기본값)
minOccurs=0, maxOccurs="unbounded"	[1..*]	1회 이상
minOccurs=0, maxOccurs=m	[n..m]	n회 이상 m회 이하

최소 발생횟수는 minOccurs, 최대 발생횟수는 maxOccurs 속성으로 표현하며, 발생횟수가 지정되지 않으면 기본값으로 1이다. UML 클래스 다이어그램으로 반복횟수를 모델링하는 방법은 반복횟수를 [초기값..종료값] 형식으로 표현한다. 다음은 반복횟수의 예와 모델링 결과이다.

```
<xsd:complexType name="Items">
  <xsd:sequnce>
   <xsd:element name="item" minOccurs="0"
     maxOccurs="unbounded" tyep="xsd:string">
   <xsd:element ref="comm" minOcucurs="0"/>
  </xsd:sequnce>
  <xsd:attributename="partNum" type="xsd:sting"/>
</xsd:complexType>
```

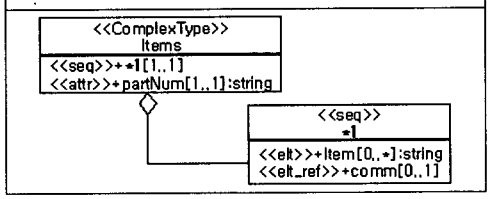

그림 12 발생횟수의 예과 모델링 결과

4. XPDL 메타모델에 대한 모델링

본 장에서는 XPDL 프로세스 메타모델를 구성하는 각 엔티티의 스키마를 UML 클래스 다이어그램으로 모델링하는 방법과 비즈니스를 수행하는 작업 단위인 액티비티 엔티티에 대한 작업 수행과정을 액티비티 다이어그램으로 모델링하는 방법을 기술한다.

4.1 XPDL 엔티티 모델링

XPDL 프로세스 메타모델의 엔티티는 XPDL 스키마로 표현된다. 다음은 XPDL 스키마를 UML 클래스 다이어그램으로 사상하기 위한 정의이다.

[정의 1] 워크플로우 프로세스 엔티티가 참조하는 하위 5 개의 엔티티에 대한 XPDL 스키마는 UML 클래스 다이어그램으로 사상된다.

4.1.1 워크플로우 프로세스 정의 엔티티

워크플로우 프로세스 정의 엔티티는 XPDL에서 <WorkflowProcess> 엘리먼트로 표현되며, 이 엘리먼트를 모델링하기 위한 규칙은 다음과 같다.

[규칙 1] 프로세스 Id와 AccessLebel은 <<attr>> 스테레오타입으로 표현하며, 워크플로우 프로세스가 참조하는 하위 5 개의 엘리먼트는 집단화 관계(◆)로 표현한다.

다음 그림은 XPDL의 워크플로우 프로세스 엘리먼트에 대하여 [규칙 1]을 적용한 결과이다.

```
<xsd:element  name="WorkflowProcess">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="xpdl:DataFields" minOccurs="0"/>
   <xsd:element ref="xpdl:Participants" minOccurs="0"/>
   <xsd:element ref="xpdl:Applications"
    minOccurs="0"/>
   <xsd:element ref="xpdl:Activities" minOccurs="0"/>
   <xsd:element ref="xpdl:Transitions" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="Id" tyep="xsd:NMTOKEN"
   use="required"/>
  <xsd:attribute name="AccessLevel"tyep="xsd:
   NMTOKEN">
 </xsd:complexType>
</xsd:element>
```

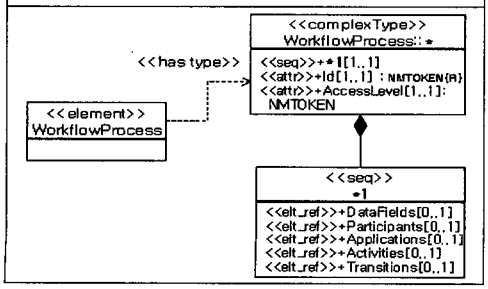

그림 13 <WorkflowProcess> 엘리먼트의 예와 모델링 결과

워크플로우 프로세스는 2 개의 속성과 프로세스 수행에 필요한 5 개의 엘리먼트를 참조하는 구문으로 되어 있다. 객체 모델링 부분에서 <<seq>> 클래스는 외부 엘리먼트를 참조하는 <<elt_ref>> 스테레오타입은 반드시 워크플로우 프로세스 엘리먼트를 구성하는 요소로 집단화 관계(◆)에 의해 표현하며, Id 속성의 {R}은 반드시 요구(required)되는 것을 의미한다.

4.1.2 워크플로우 프로세스 액티비티 엔티티

워크플로우 프로세스 액티티 엔티티는 XPDL에서 <Activity> 엘리먼트로 표현되며, 이를 모델링하기 위한 규칙은 다음과 같다.

[규칙 2] 액티비티의 Id는 <<attr>> 스테레오타입으로 표현하고, 액티비티 수행자와 제약조건 그리고 액티비티의 타입은 집단화 관계(◆)로 표현한다. 또한 여러 개의 액티비티를 포함하고 있는 <Activities>는 수행할 <Activity>에 대해서 <<execution>> 스테레오타입로 표현한다.

다음은 XPDL의 <Activity> 엘리먼트에 대하여 [규칙 2]를 적용한 결과이다.

```
<xsd:element name="Activity">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="xpdl:Description" minOccurs="0"/>
    <xsd:choice>
     <xsd:element ref="xpdl:Route"/>
     <xsd:element ref="xpdl:Implementation"/>
     <xsd:element ref="xpdl:BlockActivity"/>
    </xsd:choice>
   <xsd:element ref="xpdl:Performer" minOccurs="0"/>
   <xsd:element ref="xpdl:TransitionRestrictions"
       minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:NMTOKEN"
      use="required"/>
 </xsd:complexType>
</xsd:element>
<xsd:element name="Activities">
 <xsd:complexType>
    <xsd:element ref="xpdl:Activity" minOccurs="0"
       maxOccurs="unbounded"/>
 </xsd:complexType>
</xsd:element>
```

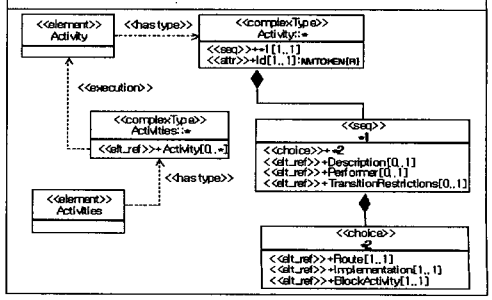

그림 14 <Activity> 엘리먼트의 예와 모델링 결과

"Activity" 엘리먼트는 비즈니스를 수행하는 작업 단위인 액티비티와 액티비티의 파리미터를 정의하고 수행에 필요한 어플리케이션과 관련 정보 등을 정의한다. 그림 14에서 <<choice>> 뒤에 사용된 식별자 "*2"는 <<seq>>에서 사용된 식별자 "*1"와 구분하기 위해 "2"를 사용했다. 액티비티의 작업 수행 과정은 4.2절에서 액티비티 다이어그램으로 표현한다.

4.1.3 워크플로우 파티시펀트 기술 엔티티

파티시펀트 엘리먼트는 액티비티의 수행자인 시스템 또는 사람을 기술하는 부분이다. 즉 파티시펀트는 액티비티 수행자를 명세하는 부분으로, XPDL에서 <Participant> 엘리먼트로 표현된다. 이를 모델링하기 위한 규칙은 다음과 같다.

[규칙 3] 파티시펀트의 Id는 <<attr>> 스테레오타입으로 표현하고, 액티비티 수행자의 타입과 설명은 집단화 관계(◆)로 표현한다. 또한 여러 개의 액티비티를 포함하고 있는 <Paticipants>는 참조할 <Participant>에 대해서 <<execution>> 스테레오타입으로 표현한다.

다음은 XPDL의 <Participant> 엘리먼트에 대하여 [규칙 3]을 적용한 결과이다.

```
<xsd:element name="Participant">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="xpdl:Description"
     minOccurs="0"/>
   <xsd:element ref="xpdl:ParticipantType"/>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:NMTOKEN"
      use="required"/>
 </xsd:complexType>
</xsd:element>
<xsd:element name="Participants">
 <xsd:complexType>
   <xsd:element ref="xpdl:Participant"
    minOccurs="0" maxOccurs="unbounded"/>
  </xsd:complexType>
</xsd:element>
```

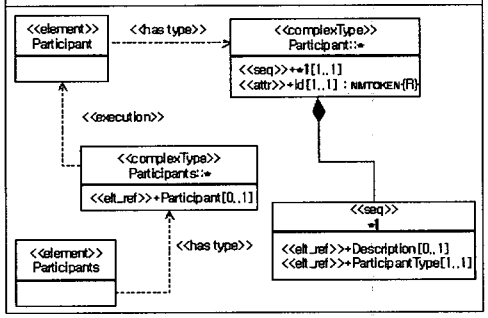

그림 15 <Participant> 엘리먼트의 예와 모델링 결과

4.1.4 트랜지션 정보 엔티티

트랜지션은 액티비티의 실행 제어조건을 기술하는 부분으로, XPDL에서 <Transition> 엘리먼트로 표현된다. 이를 모델링하기 위한 규칙은 다음과 같다.

[규칙 4] 트랜지션의 Id, 액티비티의 시작과 종료를 지정하는 "From"과 "To"는 <<attr>> 스테레오타입으로, 트랜지션의 수행 조건은 집단화 관계(◆)로 표현한다. 또한 여러 개의 트랜지션을 포함하는 <Transitions>는 수행할 <Transition>에 대해서 <<execution>> 스테레오타입으로 표현한다.

다음은 XPDL의 <Transition> 엘리먼트에 대하여 [규칙 4]를 적용한 결과이다.

```
<xsd:element name="Transition">
 <xsd:complexType>
  <xsd:sequence>
     <xsd:element ref="xpdl:Condition" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:NMTOKEN"
   use="required"/>
  <xsd:attribute name="From"type="xsd:NMTOKEN"
   use="required"/>
 <xsd:attribute name="To" type="xsd:NMTOKEN"
   use="required"/>
 </xsd:complexType>
</xsd:element>
<xsd:element name="Transitions">
  <xsd:complexType>
   <xsd:element ref="xpdl:Transition" minOccurs="0"
     maxOccurs="unbounded"/>
  </xsd:complexType>
</xsd:element>
```

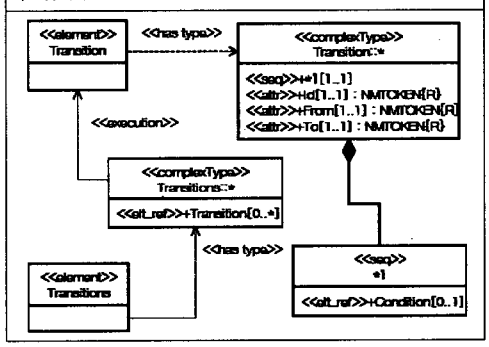

그림 16 <Transition> 엘리먼트의 예와 모델링 결과

4.1.5 워크플로우 어플리케이션 선언 엔티티

어플리케이션은 액티비티의 실행중에 참조하는 IT 어플리케이션을 선언하는 부분으로 XPDL에서 <Application> 엘리먼트로 표현된다. "Application" 엘리먼트

를 모델링하기 위한 규칙은 다음과 같다.

[규칙 5] 어플리케이션의 Id는 <<attr>> 스테레오타입으로, 액티비티에서 보낸 파라미터값을 수신 받는 파라미터와 설명은 집단화 관계(◆)로 표현한다. 또한 여러 개의 어플리케이션을 포함하는 <Applications>는 수행할 <Application>에 대해서 <<excution>> 스테레오타입으로 표현한다.

다음은 XPDL의 <Application>엘리먼트에 대하여 [규칙 5]를 적용한 결과이다.

```
<xsd:element name="Application">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="xpdl:Description" minOccurs="0"/>
   <xsd:element ref="xpdl:FormalParameters"/>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:NMTOKEN"
   use="required"/>
 </xsd:complexType>
</xsd:element>
<xsd:element name="Applications">
 <xsd:complexType>
  <xsd:element ref="xpdl:Application" minOccurs="0"
    maxOccurs="unbounded"/>
 </xsd:complexType>
</xsd:element>
```

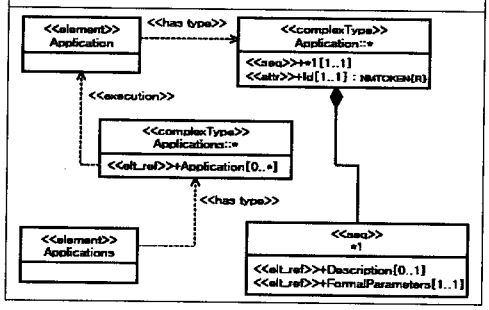

그림 17 <Application> 엘리먼트의 예와 모델링 결과

4.1.6 워크플로우 릴리번트 데이타 엔티티

릴리번트 데이타는 프로세스 수행 중에 각 프로세스 인스턴스에서 사용되거나 생성될 데이타의 타입, 초기값, 크기 등을 정의하는 부분으로, XPDL에서는 <DataField> 엘리먼트로 표현한다. 이를 모델링하기 위한 규칙은 다음과 같다.

[규칙 6] 데이타필드의 Id는 <<attr>> 스테레오타입으로, 데이타의 타입, 초기값, 크기는 집단화 관계(◆)로 표현한다. 또한 여러 개의 데이타필드를 포함하는 <DataFields>는 수행할 <DataField>에 대해서 <<execution>> 스테레오타입으로 표현한다.

다음은 XPDL의 <DataField> 엘리먼트와 이에 대하여 [규칙 6]을 적용한 결과이다.

```
<xsd:element name="DataField">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="xpdl:DataType"/>
   <xsd:element ref="xpdl:InitialValue"
minOccurs="0"/>
   <xsd:element ref="xpdl:Length" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="Id" type="xsd:NMTOKEN"
   use="required"/>
 </xsd:complexType>
</xsd:element>
<xsd:element name="DataFields">
 <xsd:complexType>
   <xsd:element ref="xpdl:DataField" minOccurs="0"
    maxOccurs="unbounded"/>
 </xsd:complexType>
</xsd:element>
```

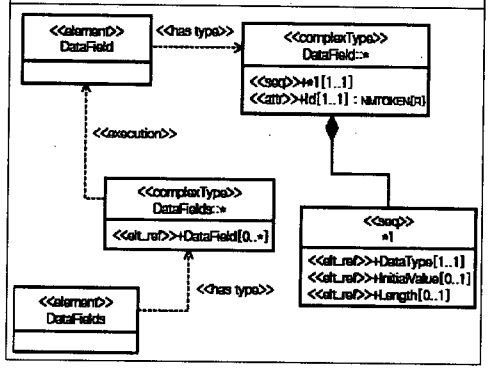

그림 18 <DataFields> 엘리먼트의 예와 모델링 결과

4.2 XPDL 액티비티 엔티티 모델링

XPDL 프로세스 메타모델의 액티비티 엔티티를 UML 액티비티 다이어그램으로 모델링을 하기 위해 다음과 같이 정의하고, XPDL의 액티비티 엘리먼트를 UML 액티비티 다이어그램으로 사상하기 위한 사상테이블을 작성하여 XPDL의 액티비티를 모델링한다.

[정의 2] 워크플로우 프로세스 액티비티 엔티티에서 액티비티와 액티비티 파라미터 그리고 제약조건은 UML 액티비티 다이어그램으로 사상된다.

4.2.1 액티비티 사상 테이블

워크플로우 프로세스 액티비티 엘리먼트에서 액티비티에 대한 표기법과 이에 대한 UML 액티비티 다이어그램으로의 사상 테이블은 다음과 같다.

표 9 XPDL 표현과 액티비티 다이어그램의 사상테이블

구 분	XPDL 표현	UML 액티비티 다이어그램	기 능
초기 노드	<Activity> <Route/> </Activity>	●	액티비티 시작
종료 노드	<Activity> <Route/> </Activity>	◉	액티비티 종료
액티비티 노드	<Activity> <Implementation/> </Activity>	▭	정규 액티비티
단일-스레드, XOR-결합	<Transitions> <Transition Id="n" From="a1" To="a2"/> </Transitions>	→	액티비티의 진행 방향
XOR-분열	<Activity> <TransitionRestriction> </Activity>	◇	조건 분기
AND-분열, AND-결합	<Activity> <Split Type="AND"> or <Join Type="AND"> </Activity>	|	동기화 막대
엔티티	<Activity> <Tool Id="n" name="Application Name"/> </Activity>	<<엔티티>> 수행될 액티비티의 집합 [수행 결과]	액티비티 수행에 필요한 엔티티

다음 절에서 이와 같은 사상테이블을 기초로 액티비티를 UML 액티비티 다이어그램으로 모델링하기 위한 상세한 방법을 기술한다.

4.2.2 액티비티 사상 규칙

(1) 단일 쓰레드(Single-thread)

하나의 액티비티가 수행된 이후 다른 하나의 액티비티로 전이는 액티비티 행위를 의미한다. 이러한 트랜지션을 모델링하기 위한 규칙은 다음과 같다.

[규칙 7] 액티비티에 대한 트랜지션은 '→' 기호로 표현한다.

다음 그림은 XPDL의 단일 쓰레드 트랜지션의 예와 [규칙 7]를 적용한 결과이다.

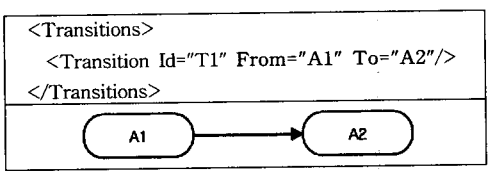

그림 19 단일 쓰레드의 예와 모델링 결과

(2) XOR-분기

하나의 액티비티가 조건에 의해 다수의 액티비티로 트랜지션되는 액티비티 행위를 의미이다. 이러한 행위를 모델링하기 위한 규칙은 다음과 같다.

[규칙 8] 액티비티에 대한 조건 분기는 '◇' 기호로 표현하고 조건식은 '[ ]' 기호로 표현한다. 또한 다수의 액티비티에 대한 분기 트랜지션은 '→' 기호로 표현한다.

다음은 XPDL에서 제공한 XOR-분기에 대한 예와 [규칙 8]를 적용한 결과이다.

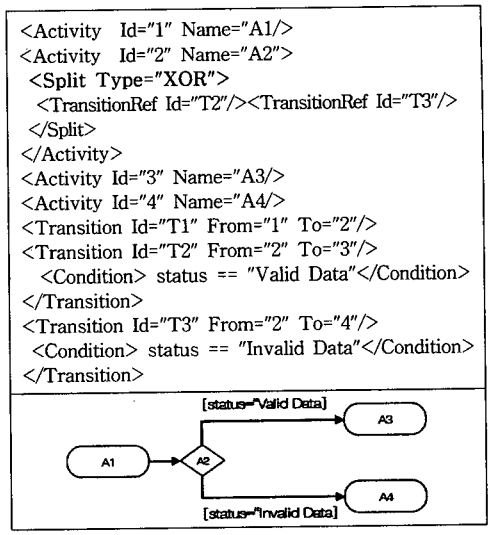

그림 20 XOR-분기의 예와 모델링 결과

(3) XOR-결합

액티비티가 조건없이 하나의 액티비티로 전이되는 액티비티 행위를 의미한다. 이러한 액티비티 행위를 모델링하기 위한 규칙은 다음과 같다.

[규칙 9] 다수의 액티비티에서 하나의 액티비티로의 트랜지션은 '→' 기호로 표현한다.

다음은 XPDL의 XOR-결합에 대한 예와 [규칙 9]를 적용한 결과이다.

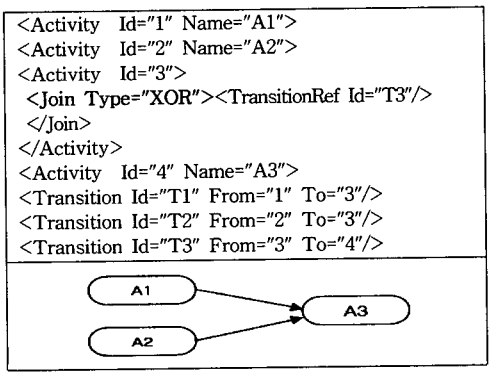

그림 21 XOR-결합의 예와 모델링 결과

(4) AND-분기

하나의 액티비티가 다수의 액티비티로 동시에 전이되는 액티비티 행위를 의미한다. 이러한 액티비티 행위를

모델링하기 위한 규칙은 다음과 같다.

[규칙 10] 하나의 액티비티와 다수의 액티비티 사이의 동기화는 ' | ' 기호로 표현하고 다수의 액티비티로의 트랜지션은 '→' 기호로 표현한다.

다음은 XPDL에서의 AND-분기에 대한 예와 [규칙 10]를 적용한 결과이다.

```
<Activity  Id="1" Name="A1"/>
<Activity  Id="2">
 <Split Type="AND">
  <TransitionRef Id="T2"/>
  <TransitionRef Id="T3"/>
 </Split>
<Activity  Id="3" Name="A2"/>
<Activity  Id="4" Name="A3"/>
<Transition Id="T1" From="1" To="2"/>
<Transition Id="T2" From="2" To="3"/>
<Transition Id="T3" From="2" To="4"/>
```

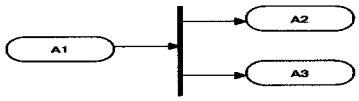

그림 22 AND-분기의 예와 모델링 결과

(5) AND-결합

다수의 액티비티가 하나의 액티비티로 전이되는 액티비티 행위를 의미한다. 이러한 액티비티 행위를 모델링하기 위한 규칙은 다음과 같다.

[규칙 11] 다수의 액티비티와 하나의 액티비티 사이는 동기화 ' | ' 기호로 표현하고 액티비티로의 트랜지션은 '→' 기호로 표현한다.

다음은 XPDL의 AND-결합에 대한 예와 [규칙 11]를 적용한 결과이다.

```
<Activity  Id="1" Name="A1"/>
<Activity  Id="2" Name="A2"/>
<Activity  Id="3">
  <Join Type="AND"><TransitionRef Id="T3"/>
  </Join>
<Activity  Id="4" Name="A3"/>
<Transition Id="T1" From="1" To="3"/>
<Transition Id="T2" From="2" To="3"/>
<Transition Id="T3" From="3" To="4"/>
```

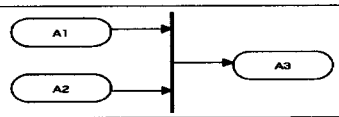

그림 23 AND-결합의 예와 모델링 결과

(6) 엔티티

액티비티가 수행 도중에 참조하는 엔티티를 의미한다. 여기서 엔티티는 클래스 다이어그램으로 모델링되며 모델링 규칙은 다음과 같다.

[규칙 12] 클래스 이름 부분에 참조하는 엔티티를 <<application>> 스테레오 타입으로 표현하고, 수행되는 엔티티 이름은 '밑줄'로 표현한다. 또한 엔티티에 대한 수행결과는 '[ ]' 기호로 표현한다.

다음 예문은 XPDL에서 제공한 엔티티 예와 [규칙 12]를 적용한 결과이다.

```
<Activity Id="1" Name="Set Credit Info">
  <Implementation>
<Tool Id="setCreaditInfo" Type="APPLICATION">
 <ActualPrameter>accountNum </ActualPrameter>
 <ActualPrameter> amount </ActualPrameter>
 <ActualPrameter> cardType </ActualPrameter>
 <ActualPrameter> creditInfo </ActualPrameter>
</Tool>
<Application Id="setCreditInfo">
```

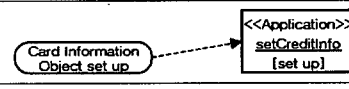

그림 24 엔티티의 예와 모델링 결과

5. XPDL 워크플로우 모델링 적용

이 장에서는 4장에서 제안한 XPDL 스키마에 대한 사상기법을 검증하기 위하여, "신용카드 상태체크 시스템"의 워크플로우를 모델링하는데 제안된 기법을 적용한다. 즉 모델링 대상은 온라인 상에서 사용자가 카드로 물품을 주문했을 경우 사용자의 신용 카드 상태를 체크하는 시스템으로, 이 시스템에 대한 워크플로우를 모델링하여 본 논문에서 제안한 방법의 유효성을 검증한다.

5.1 카드 상태체크 시스템 구성도

본 논문에서의 모델링 대상인 카드 상태체크 시스템의 워크플로우는 다음 그림과 같다.

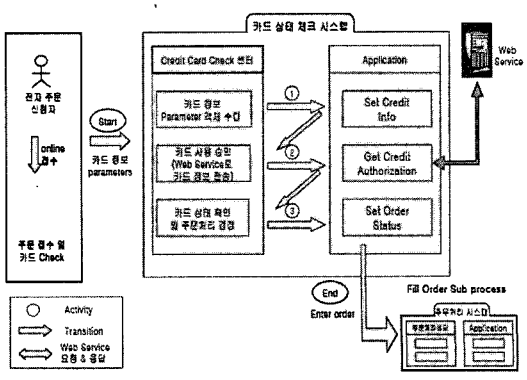

그림 25 카드 상태체크 워크플로우

```
<?xml version="1.0" encoding="us-ascii" ?>
<Package>   //3개의 워크플로우 프로세스 정의
 <WorkflowProcesses>
  <WorkflowProcess id=1  Name="Eorder"/>  // 물품 주문
  <WorkflowProcess id=3  Name="Fillorder"/> // 주문 처리

 <WorkflowProcess id=2 Name="CreditCheck"> //카드 체크
  <Applications>   //어플리케이션 선언부
   <Application  Id="setCreditInfo">
    <Description> 카드 정보의 객체를 생성 </Description>
    <FormalParameters>
     <FormalParameter Id="accountNumer" index="1" Mode="IN"/>
     <FormalParameter Id="amount" index="2" Mode="IN"/>
     <FormalParameter Id="cardType" index="3" Mode="IN"/>
     <FormalParameter Id="creditInfo" index="4" Mode="OUT"/>
    </FormalParameters>
   </Application>
   <Application  Id="getCreditAuthorization">
    <Description> 카드 인증 수행 </Description>
    <ExternalReference location="http://wfmc.org/standards/docs/
     xpdl_sample/creditService.wsdl" xref="GetCreditAuthorization>
   </Application>
   <Application  Id="setOrderStatus">
    <Description>카드 상태에 따른 주문처리 결정</Description>
    <FormalParameters>
     <FormalParameter Id="creditStatus"index="1" Mode="IN"/>
     <FormalParameter Id="orderStatus" index="2"
      Mode="OUT"/>
    </FormalParameters>
   </Application>
 <Applications/>

 <Participants> //액티비티 수행자 명세 부분
   <Participant id = "DBConnection">
     <ParticipantType = "SYSTME"/>
        <Description>DB 자원 참조</Description>
   </Participant>
</Participants>

 <Activities> //액티비티 정의부분
  <Activity Id="start" Name="TransformData">
   <Discription>주문정보를 카드 체크 시스템으로 전송
   </Discription>
   <Implementation>
     <Tool Id="transformData" Type="application">
       <ActualParameters>
        <ActualParameter>orderString </ActualParameter>
        <ActualParameter>orderInfo </ActualParameter>
       </ActualParameters>
     </Tool>
   </Implementation>
  </Activity>
  <Activity Id="1" Name="SetCreditInfo">
   <Discription>카드정보의 객체 생성 </Discription>
   <Implementation>
    <Tool Id="setCreditInfo" Type="apllication">
     <ActualParameters>
       <ActualParameter>accountNumber</ActualParameter>
       <ActualParameter>amount </ActualPrameter>
       <ActualParameter>cardType </ActualParameter>
       <ActualParameter>creditInfo </ActualParameter>
      </ActualParameters>
     </Tool>
    </Implementation>
    // 생성된 카드 정보를 데이터베이스에 저장.
    <Performer> DBConnection </Performer>
   </Activity>
   <Activity Id="2" Name="GetCreditAuthorization">
    <Discription>카드정보를 getCreaditAuthorization의 어플
    리케이션으로 카드 인증을 수행</Discription>
    <Implementation>
     <Tool Id="getCreditAuthorization" Type="application">
      <ActualParameters>
       <ActualParameter>creditInfo </ActualParameter>
       <ActualParameter>creditStatus </ActualParameter>
      </ActualParameters>
     </Tool>
    </Implementation>
   </Activity>
   <Activity Id="3" Name="SetOrderStatus">
    <Discription>카드정보를 setOrderStatus으로 카드 상태를
    확인하여 주문처리 결정. </Discription>
     <Implementation>
      <Tool Id="setOrderStatus" Type="application">
       <ActualParameters>
        <ActualParameter>creditStatus</ActualParameter>
        <ActualParameter>status </ActualParameter>
       </ActualParameters>
      </Tool>
     </Implementation>
    </Activity>
    <Activity Id="end" Name="EnterOrder">
     <Discription>사용자가 주문한 물품을 처리하기 위해
      주문 처리 시스템으로 작업을 전이. </Discription>
     <Implementation>
      <Tool Id="enterOrder" Type="application">
       <ActualParameters>
        <ActualParameter>orderInfo </ActualParameter>
        <ActualParameter>orderNumber</ActualParameter>
       </ActualParameters>
      </Tool>
     </Implementation>
    // 데이터베이스에 저장된 카드정보를 가져옴
     <Performer>DBConnection </Performer>
    </Activity>
   </Activities>

  <Transitions>  //액티비티 실행 제어 정보
    <Description>액티비티들의 실행 제어조건 </Description>
    <Transition  Id="1"  Form="start"  To="1"/>
    <Transition  Id="2"  Form="1"  To="2"/>
    <Transition  Id="3"  Form="2"  To="3"/>
    <Transition  Id="4"  Form="3"  To="end"/>
   </Transitions>
  </WorkflowProcess>
 </WorkflowProcesses>
</Package>
```

그림 26 카드 상태체크 워크플로우에 대한 XPDL 문서

사용자가 온라인 상에서 물품을 주문하고 사용자의 카드로 대금을 결재하는 과정에서 사용자의 카드 상태를 점검하여 이상이 없으면 카드 결재가 이루어지는 시스템이다. 이 시스템의 작업흐름은 먼저 사용자의 물품 주문과 사용자의 카드 정보를 파라미터로 입력받는 Start 액티비트에서 시작된다. 실제 카드 상태 체크를 수행하는 액티비티 ①에서 파리미터를 받아 사용자의 카드 정보에 대한 객체를 생성한 다음, 액티비티 ②에서 사용자의 카드를 인증받기 위해 인증 서버에 카드 정보를 전송한다. 또한 액티비티 ③에서는 카드 상태에 문제가 없는가를 확인하여 주문처리를 결정한다. 끝으로 실제 사용자가 주문한 물품을 처리하기 위해 사용자의 주문 정보를 주문처리 시스템으로 전송하는 작업은 End 액티비티에 의해서 수행된다.

5.2 카드 상태체크 시스템의 워크플로우에 대한 XPDL 문서

그림 25의 "카드 상태체크 시스템"에 대한 워크플로우를 XPDL을 이용하여 표현하면 앞의 그림 26과 같다.

5.3 XPDL 워크플로우 모델링

5.2절에서 XPDL을 이용하여 표현한 "카드 상태체크 시스템"에 대한 워크플로우를 UML 클래스 다이어그램과 액티비티 다이어그램으로 모델링한 결과를 기술한다.

5.3.1 엔티티 객체 모델링

5.2절의 워크플로우에서 엔티티들을 UML 클래스 다이어그램으로 모델링한 결과는 그림 27과 같다.

위의 그림을 살펴보면 워크플로우 팩키지 안에 동일 영역에서 수행되는 워크플로우 프로세스(Eorder, CreditCheck, Fillorder)들이 클래스 다이어그램으로 표현되어 있다. 이들 중 카드 상태를 점검하는 CreditCheck 프로세스에 대한 Activitys, Applications, Transitions, Participants 엔티티를 클래스 다이어그램으로 모델링하였다. 여기서 Activitys에는 프로세스를 수행하는 액티비티들이 정의되어 있으며, Transitions은 이러한 액티비티들의 작업 순서를 제어하는 역할을 담당한다. 그리고 Apllications은 시스템에서 필요한 IT 애플리케이션을 정의하여 액티비티들이 호출하여 사용할 수 있도록 하며, Participants는 DBConnection 시스템을 명세하여 데이타베이스 관련 액티비티들에 의해서 실행된다.

5.3.2 액티비티 모델링

"카드 상태체크 시스템" 내의 프로세스 액티비티 대하여 UML 액티비티 다이어그램으로 모델링한 결과는 그림 28과 같다.

프로세스 행위를 단계적으로 기술하면, 먼저 물품을 주문하는 사용자의 정보(주문 및 카드정보)를 카드 상태체크 센터(Card Check Chenter)로 전송하는 것으로 시작으로 한다. 첫 번째 단계에서 Card Information Object Set Up 액티비티에 의해서 시스템에서 사용할 수 있는 카드정보 객체를 생성한다. 두 번째 단계는 액티비티의 분할이 발생한다. DBConnection 액티비티에서는 생성된 객체정보를 데이타베이스에 저장하고, Get Creadit Authorization 액티비티에서는 카드 상태를 점검하기 위해 지정된 서버로 카드 정보를 전송한다. 세 번째

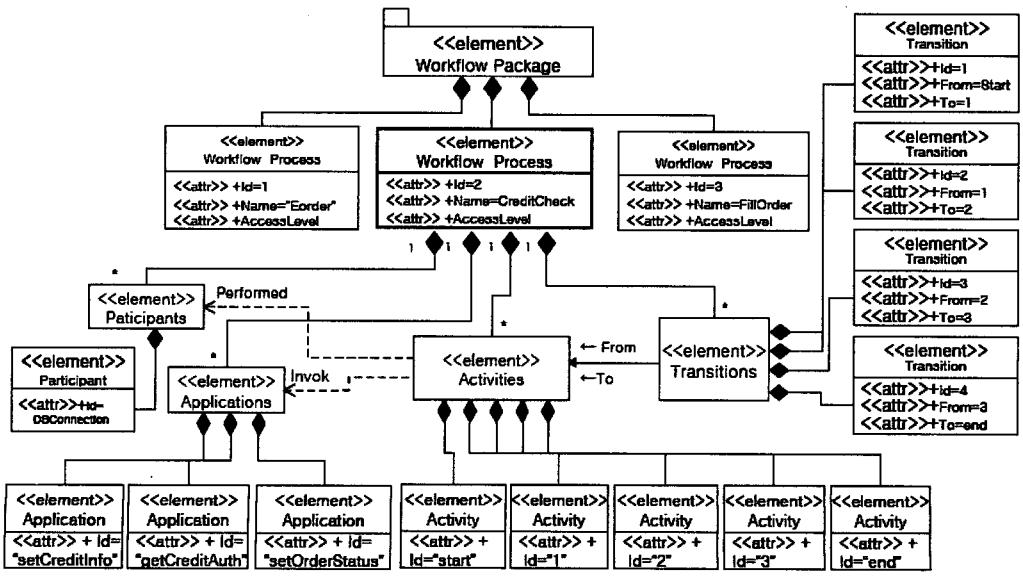

그림 27 카드 상태체크 시스템의 워크플로우 엔티티에 대한 모델링 결과

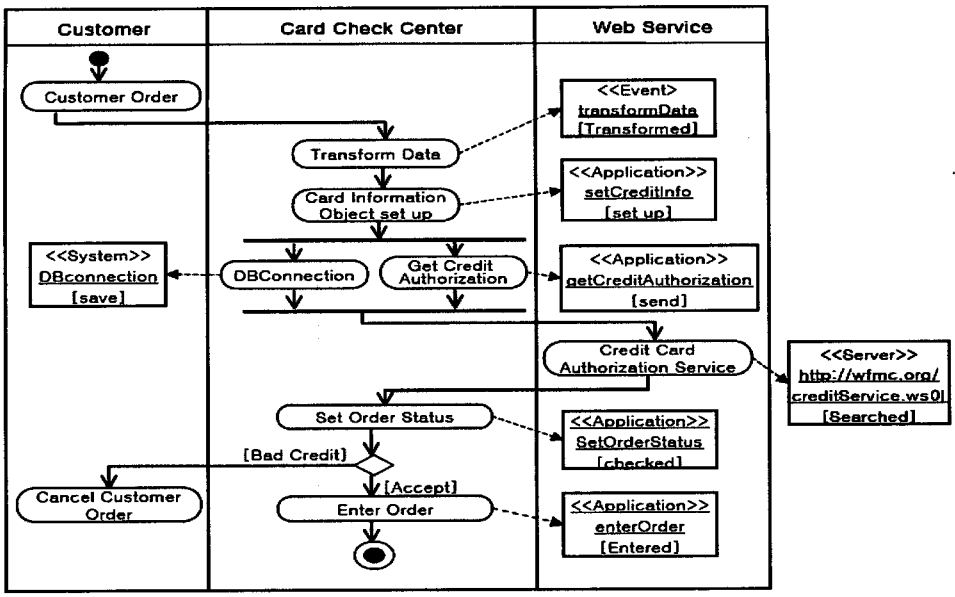

그림 28 카드 상태체크 시스템의 프로세스 액티비티에 대한 모델링 결과

단계는 카드 상태체크 서버에서 사용자의 카드 상태를 점검하는 액티비티가 수행된다. 네 번째 단계는 Set Order Status 액티비티가 카드 상태에 문제가 없는가를 확인하여 주문처리를 결정한다. 만약 이 과정에서 사용자의 카드 사용에 문제(Bad Credit)가 있으면 Cancel Customer Order 액티비티에 의해서 고객의 물품 주문은 취소된다.

6. 결론 및 향후 연구 과제

기업간의 e-비즈니스 환경은 기업 내부의 업무 표준화와 통합을 기반으로, 기업 간의 상호 교류를 통한 협업으로 발전되고 있다. 따라서 많은 기업들이 고객과 공급자 그리고 비즈니스 파트너의 정보를 공유하기 위한 일련의 비즈니스 프로세스를 수행한다. 그러므로 기업간의 상이한 문서 양식과 프로세스에 대한 표준화는 전자상거래의 효율성과 생산성을 극대화시킬 수 있다. 이러한 비즈니스 프로세스의 표준화를 위한 대표적인 기술이 워크플로우이다. 기업내 워크플로우의 표준화 정도는 기업간의 협업과 거래 프로세스의 구축에 중요한 요인이 되고 있다.

본 논문은 기업간 워크플로우의 통합 시스템을 구축하기 위해 필요한 핵심 요소인 워크플로 프로세스 정의 및 교환을 UML 다이어그램으로 모델링하여 기업간의 상호 연동과 협업을 위한 업무흐름 파악을 용이하게 하기 위한 모델링 방안을 제안하였다. 이를 위해 XPDL 프로세스 메타모델을 UML 다이어그램으로 모델링하기 위한 사상규칙을 정의하고, 이를 기반으로 엔티티와 액티비티를 모델링하였다. 그리고 제안된 기법을 "카드 상태체크 시스템"에 대한 워크플로우를 모델하는데 적용하여 제안된 기법의 유효성을 검증하였다. 제안된 모델링 기법은 UML의 직관적인 표현법과 사상규칙을 적용한 정형화된 표현으로 인하여 워크플로우 프로세스 모델링의 전 분야에 적용될 수 있을 것으로 기대된다.

향후 연구 과제는 워크플로우 모델링을 위한 사상규칙을 발전시키고, 이를 위한 자동화 도구를 개발하여 워크플로우 모델링의 생산성 및 경제성을 높일 수 있는 연구가 후행되어야 한다.

참 고 문 헌

[1] 신상철, "기업간 워크플로우 통합 기술 표준 연구 보고서", 한국 전산원, 2002.

[2] D. Jutla, et al., "Making Business Sense of Electronic Commerce," IEEE Computer, Vol. 32, pp. 67-75, 1999.

[3] 원재강, 김학성, 이문영, 김광훈, 정관희, "워크플로우 표준화 동향 분석", 한국인터넷정보학회, 춘계 학술 발표 논문집 제1권 1호, 2000.

[4] 김광훈, "워크플로우 기술 I", 한국정보통신기술협회(TTA) 저널, 85호, pp. 107-118, 2003.

[5] Nicholas Routledge, Linda Bird, and Andrew Goodchild, "UML and XML Schema," Australasian Database Conference (ADC2002), Vol. 5, pp. 157-166, 2002.

[6] Dave Carlson, "Modeling XML Vocabularies with UML: Part I~III", http://www.xml.com/pub/a/2001/

10/10/uml.html, Oct. 2001.
[7] XMLmodeling.com "UML Models of W3C XML Schema," http://www.xmlmodeling.com/models/w3c_xsd/v1.0/index.html, Nov. 2004.
[8] Ricardo M. Bastos, Duncan Dubugras A, "Extending UML Activity Diagram for Workflow Modeling in Production Systems," Hawaii International Conference on System Sciences (HICSS'02), Vol 9, pp. 291-301, 2002.
[9] Ping Jiang, Quentin Mair, and Julian Newman, "Using UML to Design Distributed Coolaborative Workflow: from UML to XPDL," Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03), pp. 71-77, 2003.
[10] 왕보, 김재정, 유철중, 장옥배, "XPDL 문서 생성을 위한 UML 액티비티 다이어그램의 확장", 한국정보과학회, '03가을 논문집(2), pp. 247-249, 2003.
[11] Grady Booch, Ivar Jacobson, Jim Rumbaugh, "OMG Unified Modeling Language Specification," http://www.omg.org/docs/ formal/03-03-01.pdf, Mar. 2003.
[12] Workflow Process Definition Interface - XML Process Definition Language(XPDL), WfMC, http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf, Oct. 2002.
[13] The Workflow Reference Model(WFMC-TC-1003), http://www.wfmc.org/standards/model.htm, Jan. 1995.
[14] Henry S. Thompson, David Beech, Noah Mendelsohn, and Murray Maloney, "XML Schema Part 1: Structures," W3C Recommendationhttp://www.w3.org/TR/xmlschema-1, Oct. 2004.
[15] Erwan Breton, Jean Bezivin, "Weaving Definition and Execution Aspects of Process Meta-Models," Hawaii International Conference on System Sciences(HICSS'02), Vol 9, pp. 290-300, 2002.
[16] XPDL Sample, http://www.wfmc.org/standards/docs/xpdl_sample/sample %20workflow%20process.xpdl
[17] 고현민, 손명근, 오윤주, 배두환 "UML Activity Diagram을 통한 비즈니스 프로세스 모델링 가능성 분석", 한국정보과학회, '03봄 논문집(B), pp. 112-114, 2003.

김 진 성

1992년 원광대학교 신문방송학과(사회과학사). 2002년 전북대학교 컴퓨터 정보(이학석사) 2003년~현재 전북대학교 컴퓨터 통계 정보학과 박사과정. 관심분야는 XML, 객체지향 모델링, 워크플로, W3C 웹 서비스, PKI 등

유 춘 식

1991년 전북대학교 전산통계학과(이학사). 1994년 전북대학교 전산통계학과(이학석사). 2005년 전북대학교 전산통계학과(이학박사). 관심분야는 XML, 스키마 통합, 유비쿼터스, 적응형 사용자 인터페이스, W3C 웹 서비스 등

김 용 성

1978년 고려대학교 수학과(이학사). 1984년 광운대학교 전산학과(이학석사). 1992년 광운대학교 전산학과(이학박사). 1985년~현재 전북대학교 전자정보공학부 교수. 1996년~1998년 1월 한국학술진흥재단 전문위원. 관심분야는 XML, 사용자 중심의 정보검색, 다중 사용자 인터페이스, W3C 웹 서비스 등