한국경영과학회/대한산업공학회 2003 춘계공동학술대회
2003 년 5 월 16 일-17 일 한동대학교(포항)

비즈니스 프로세스 자동화를 위한 실행 모델의 설계

Design of an enactment model for business process automation

허원창[1††], 배준수[2], 강석호[1]

서울대학교[1] / 성결대학교[2]

†† 연락저자: 151-742, 서울시 관악구 신림9동 서울대학교 공과대학 산업공학과,
Tel. 02-880-7360, E-mail. bethoven@ara.snu.ac.kr

Abstract

본 연구는 비즈니스 프로세스의 자동화된 실행 및 관련 데이터의 관리를 지원하는 BPMS (Business Process Management System)를 위한 프로세스 실행 모델을 제안한다. BPMS의 효과적 개발을 위해서는 프로세스의 내용을 표현하는 프로세스 정의 모델이 필요함과 동시에, 네트워크 상에서 그것이 해석되고 실행되는 방식을 규정하기 위한 프로세스 실행 모델도 필요하다. 이는 프로세스에 관여되는 분산된 개체들간의 협력적 의사 소통을 효과적으로 지원할 수 있고, 이를 통해 시스템간의 상호운용성을 고양하고 프로세스 모델이 갖는 경직성을 극복할 수 있는 BPMS의 효과적 개발의 바탕이 될 수 있다.

Keywords: 비즈니스 프로세스, BPMS

1. 서론

일반적으로, 비즈니스 프로세스의 자동화 및 관리의 개념은 비단 근래 활발해진 기업간 거래를 위해서만 사용되는 개념은 아니었다. 기업 내부에 정보시스템이 도입되기 시작한 시점부터 이미 그룹웨어나 워크플로우(Workflow)의 개념은 존재하였고, 문서의 결제 위주로 진행되는 정형화되고 반복적인 업무 프로세스의 자동화를 위한 시스템도 활발히 사용되고 있었다.

근래에, 인터넷이라는 개방적 통신 인프라의 폭발적인 발달과 더불어, 기업간의 업무 교류가 활발해 지면서 기업내부의 업무 프로세스들은 어떤 형태로든 외부의 다른 기업의 업무 프로세스와 얽혀지게 되었다. 이에 따라, 과거의 프로세스 자동화의 개념은 이제 어떻게 기업간에 얽혀있는 업무 프로세스의 흐름을 자동화하고 관련 정보들을 관리할 것인가에 초점을 맞추는 방향으로 변모하고 있다.

이러한 기업간 협력적 프로세스 관리를 지원하는 시스템의 설계 및 개발을 위해서는 정형화된 업무 절차의 정의 및 표현을 위한 프로세스 모델과 프로세스에 정의된 절차에 따른 업무의 할당 및 진행을 방식을 담당하는 실행 모델의 두 가지 모델이 필요하다. 하지만, 현재 많은 단체에서 기업간 거래 프로세스의 표준화를 위해 다양한 형태의 프로세스 모델을 제안하고 있는데 비하여, 이러한 프로세스 모델을 적용하고 실행할 수 있도록 지원하는 시스템에 적용 가능한 구체적인 실행 모델은 제안되고 있지 않다.

따라서, 대부분의 BPMS들에서 프로세스의 실행이 구현하는 방식은 개별 시스템이 사용하는 프로세스 모델과 개발 환경, 통신 인프라 등에 종속적이게 된다. 이는 기업간 이형적 프로세스간의 통합적 지원이 어렵고, 프로세스 모델의 경직성, 즉 실행 시점의 상황 변화를 반영하기 힘들며, 참여자들 간의 다양한 형태의 협업(Collaboration)을 어렵게 한다.

본 연구는 프로세스의 정의 및 표현과 관련되는 프로세스 모델과는 별도로, 프로세스의 실행 및 관리 방식을 규정하는 실행 모델을 제안한다. 이러한 실행 모델은 기업간 거래자동화를 효과적으로 지원할 수 있는 BPMS의 설계 및 개발을 위한 중심 모델로서 이용될 수 있을 것이다.

2. 프로세스 모델과 실행 모델

근래, 업무 프로세스, 특히 기업간 거래 프로세스의 자동화 및 관리에 대한 관심이 증대되면서 여러 단체들로부터 다양한 프로세스 모델들이 제안되고 있다. 본 연구에서는 이를 크게 워크플로우, e-business, 그리고 웹 서비스(Web Service)의 세 가지 적용분야로 나누어 살펴보도록 한다. 이와 더불어, 이러한 프로세스 모델의 실행을 위한 기반 구조(infrastructure)로서 코디네이션 모델들을 살펴보도록 하겠다.

2.1 프로세스 모델

(1) 워크플로우

주로, 기업 내부의 반복적이며 정규화된 업무프로세스의 자동화를 담당하는 워크플로우 관리스시템(WFMS)에서 업무프로세스를 정의하기 위한 프로세스 모델로서 WFMC (Workflow Management Coalition)에서 제안한 XPDL (XML Process Definition Language)이 있다. XPDL 은 XML로 기술된 언어로써, 프로세스를 구성하는 업무들(Activities)와 이들간의 선후관계(Transitions)를 각각 노드(Node)와 에지(Edge)로 구성된 그래프 형태로 표현하도록 되어 있다. 또한, 업무간의 선후관계를 정의하는 데 있어서 업무의 진행 방향(Routing)을 결정하는 제약조건을 설정할 수 있고, 여러 개의 업무를 묶어 업무 집합(Activity Set)형태로 표현할 수도 있다. ()

(2) e-business

e-business를 위한 프로세스 모델로는 ebXML의 BPSS(Business Process Specification Schema)와 RosettaNet의 PIP(Partner Interface Process)이 가장 대표적이다. 이들은 e-business를 위한 표준화된 비즈니스 프로세스를 명세하기 위해 사용되는 모델링 언어이다. 이들은 워크플로우와는 달리 기업간의 거래상에서 발생하는 비즈니스 트랜잭션(Transaction)과 그 구성법(Choreography)에 초점을 맞추어 원활한 e-business를 표현할 수 있도록 지원하는데 초점을 맞추고 있다. BPSS는 UML형태와 XML형태의 스키마로 제공되어 협력적 업무프로세스를 표현할 수 있도록 하고 있고, PIP은 기업간 거래의 각 세부 기능 영역 및 단계에서 발생하는 업무 절차 및 데이터 메시지의 내용 및 구조를 자세하게 정의하고 있다.

(3) 웹 서비스(Web Service)

웹 서비스가 e-business를 위한 효과적인 인프라로 대두되면서, 이를 비즈니스 프로세스를 구성하는 주요 요소로 기업간 거래 프로세스를 정의할 수 있도록 지원하는 언어들이 등장하고 있다. 이러한 언어들은 기존의 웹 서비스들을 조합하고 조율하여 프로세스 형태로 엮어낼 수 있도록 하는데 초점을 맞추고 있다. 기본적으로, 이들은 개별 업무를 정의하는 WSDL (Web Service Definition Language)를 기반으로 이를 확장한 형태로 표현되는데, 이러한 형태의 언어로 대표적인 것들이 BPMI(Business Process management Initiative)의 BPML(Business Process Modeling Language), IBM의 WFSL(Web Service Flow Language)와 Microsoft의 XLANG(XML business process LANGuage)을 통합하여 만든 BPEL4WS(Business Process Execution Language for Web Services), 그리고 Sun Microsystems를 중심으로 한 WSCI(Web Service Choreography Interface) 등이 있다.

2.2 실행 모델

본 연구에서 소개할 실행 모델들은 원래 분산시스템 분야에서 자율적(Autonomous)인 에이전트(Agent)들간의 원활한 데이터 교환을 지원하기 위해 제안되어 사용되어 왔던 프로세스 코디네이션 모델이다.

이는 크게 공유된 저장 공간을 기반으로 한 모델과, 각 이벤트의 발생에 기반한 모델로 구분된다. 전자에서는, 각 에이전트가 공유된 저장공간에 투플(tuple)이라 불리는 단위 데이터를 생성, 삭제하거나, 특정 투플이 그 공간에 존재하는 지를 확인하고 읽어옴으로써 서로간에 의사소통을 하는 방식이다. 반면에, 이벤트 기반 모델에서는, 각 에이전트가 활동 시 주요 시점마다 특정한 이벤트를 발생시키고 이러한 이벤트를 필요로 하는 에이전트가 이를 소모함으로써 서로간에 통신하게 되는 방식이다.

그림 [1]은 공유된 저장 공간을 두 프로세스 P1과 P2가 통신하는 메커니즘을 간략하게 도식화 한 것이다.

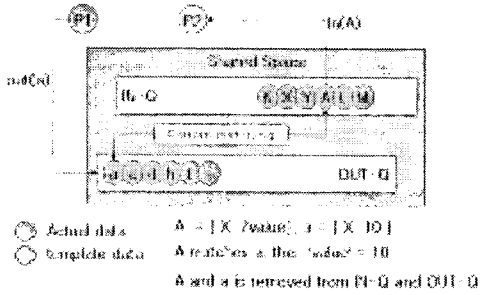

그림 [1]. 공유 저장 공간을 통한 코디네이션

그림에서 P2는 공유 저장 공간에 A라는 투플을 in(A) operation을 통해 저장한다. 이는 P2가 투플 A를 통해 변수 X의 값을 알고 싶다는 의사를 표현한 것으로, 이후 이와 부합(match)하는 데이터가 저장될 경우 그 값을 P2가 전달 받게 되는 것을 의미한다. 한편, P1은 어떤 연산의 결과로 생성된 투플 a를 out(a) operation을 통해 저장공간에 저장한다. 투플 a는 투플 A와 데이터 구조가 부합됨으로써, P2는 P1으로부터 변수 X의 값이 10이라는 정보를 얻게 된다.

이러한 형태의 메커니즘이 갖는 여러 가지 특징들 중 주목할만한 것은 참여하는 에이전트들이 서로 시간적, 공간적으로 분리될 수 있다는 점이다. 즉, 에이전트들이 대화를 위하여 서로를 명시적으로 명명(naming)할 필요가 없으며,

또한 시간적으로도 동기화 될 필요가 없음을 의미한다. 이러한 특징은, 에이전트들의 자율성과 독립성을 충분히 보장하면서도, 서로간의 협력적인 업무 달성을 가능하게 한다.

일반적으로 공유된 저장공간은 데이터나 이벤트의 입출력을 위한 인터페이스만을 제공하고 프로세스의 진행 및 관리와 관련된 서비스는 제공하지 않는다. 하지만, 프로세스의 관리 및 업무 할당을 담당하는 특별한 에이전트를 포함시킴으로써 이러한 일들이 가능하게 될 수 있다. 이러한 메커니즘이 잘 적용될 수 있는 응용시스템으로는 경매시스템, e-marketplace, 또는 여러 형태의 모니터링 시스템 등이 있다.

3. Data-associative enactment model

개념적인 수준에서 바라본 BPMS(Business Process Management System)의 구조는, 그림 [2]에 도식화한 것 같이, 서로 인터페이스 하는 두 가지 모델, 즉 정의 모델(Definition model)과 그 실행 모델(Enactment model)로 구분하여 표현될 수 있다.

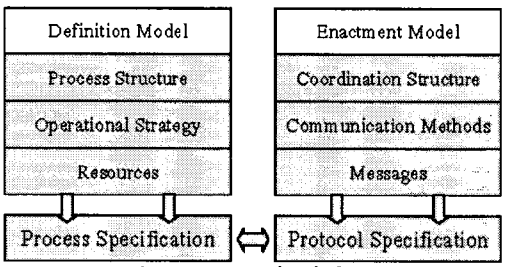

그림 [2]. BPMS의 개념 구조

정의 모델은 비즈니스 프로세스의 구조(Structure), 진행 규칙(Operation strategy) 및 데이터나 구성원들과 같은 자원(Resources)를 명세하고, 그 실행 모델은 프로세스의 실행을 위해 사용되는 구성원간의 통신 방법을 정의하는 코디네이션 메커니즘(Coordination Structure)와, 이 메커니즘 하에서 개별 구성원들이 서로간의 데이터 교환 및 서비스 호출을 위해 사용하는 통신 메소드 (Communication methods), 그리고 이러한 메소드를 호출하는 데 필요한 입출력 데이터를 정의하는 메시지(Message)의 정의로 구성된다. 이러한 개별적인 개념 모델의 정립을 바탕으로 프로세스 정의를 위한 Process Specification과 실행을 위한 Protocol Specification을 만들어 낼 수 있다.

그림 [3]은 공유 저장공간을 이용한 코디네이션 모델을 사용하여 프로세스 실행 모델을 추상화하여 도시한 것이다.

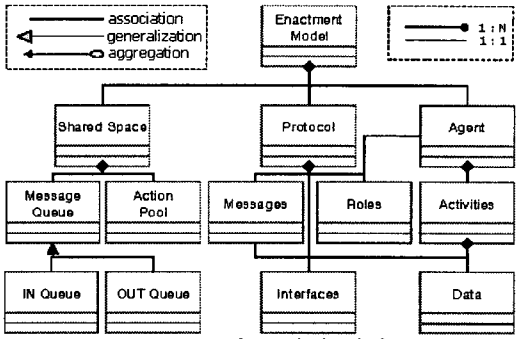

그림 [3]. 프로세스 실행 개념 모델

프로세스의 명세에 관해서는 앞서 살펴보았듯이 현재 다양한 모델들이 제안되고 있기 때문에, 본 연구에서는 비즈니스 프로세스의 실행(Enactment)과 관련된 모델의 정립에 초점을 맞추었다.

그림에서, 프로세스 실행 모델을 구성하는 개체(Entity)로는 크게, 프로세스 코디네이션을 관장하는 공유 데이터공간(Shared Space)과, 이를 활용하는 통신 구조를 정의하는 프로토콜(protocol), 그리고 프로세스에 참여하는 구성원과 자원들을 표현하는 에이전트(Agent)로 구성된다. 데이터 공간에는 데이터와 이벤트를 저장하는 대기행렬(Queue)과 이러한 메시지들을 처리하는 메시지 처리기가 정의되며, 프로토콜에는 구성원들이 통신을 위해 사용하는 인터페이스와 이를 사용하기 위해 필요한 입출력 메시지, 그리고 프로토콜 내에서 각 구성원이 수행하게 되는 역할을 정의하게 된다. 마지막으로, 에이전트는 프로세스에 참여하는 자원들과 그들의 역할, 그리고 그들이 담당하는 업무들과 관련 데이터를 명세하게 된다. 이러한 데이터 개체들은 실행모델에 독립적으로 존재하는 것도 있지만, 역할, 업무, 데이터와 같은 개체는 프로세스 정의모델에 속하는 개체와 연관관계를 맺어야 한다.

3.1 DAM (Data associative middleware)

본 연구의 프로세스 실행 모델을 구성하는 가장 핵심적인 구성요소로서 DAM (data-associative middleware)이 있다. DAM은 프로세스에 참여하는 에이전트간의 데이터 교환 및 의사 소통을 매개하는 공유된 데이터 공간을 구현한 것이다. 그 기본 구조는 그림 [4]와 같다.

그림에서 보는 바와 같이 DAM은 두 개의 메시지 대기행렬(Message Queue)과, 연관화 함수(Association function), 그리고 메시지 처리기(Message Handler)로 구성된다. 각 구성요소에

대하여 좀더 자세하게 살펴본다.

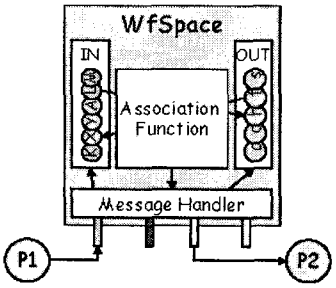

그림 [4]. Data-associative Middleware

메시지 대기행렬에는 publish 혹은 subscribe 오퍼레이션을 통해 발생된 데이터나 메시지를 저장하는 역할을 하고, 연관화 함수는 저장된 메시지들 간의 연관관계를 판별하는 역할을 한다. 예를 들면, subscribe된 특정 메시지에 대하여 이와 매칭되는 publish 메시지를 찾아내는 일이다. 메시지 처리기는 에이전트들이 발생한 이벤트나 데이터를 받아 대기행렬에 삽입하거나, 연관된 메시지가 발견되었을 때, 이를 해당 에이전트에게 통보하는 기능을 한다.

A. 메시지 처리기 (Message handler)

메시지 처리기는 DAM을 통해 교환되는 모든 메시지의 수신, 저장, 전송, 삭제 및 라우팅(routing)을 담당한다. 메시지 처리기의 처리 절차는 그림 [5]에 개념적으로 도식화 하였다.

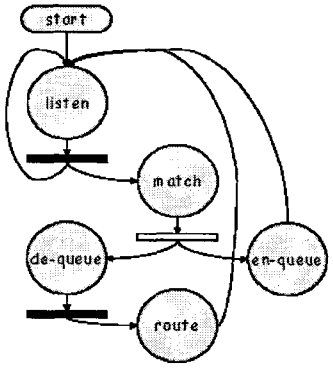

그림 [5] Message handling procedure

그림에서 보는 바와 같이, 메시지 처리기의 활동은 우선 메시지를 전송 받을 수 있는 여러 개의 네트워크 포트를 열고 해당 포트에서 메시지를 감시(listening)하는 상태로부터 시작된다. 특정 포트에 메시지가 도착하면 도착한 메시지를 이용하여 대기 행렬에 존재하는 메시지들에 대한 연관성 판별(matching) 단계를 진행시키고, 병렬적으로 다시 감시 단계로 복귀한다.

연관성 판별 단계에서는 DAM에서 제공되는 여러 가지 연관화 함수들 중 도착한 메시지의 타입에 맞는 함수를 이용하여, 도착 메시지와 반대 종류의 메시지 대기행렬에 있는 메시지들을 순차적으로 검색한다.

연관성이 있는 메시지가 발견되었을 경우 해당 메시지를 제거하는 제거(Dequeue) 단계로 진행하고, 그렇지 않을 경우에는 도착 메시지를 삽입하는 삽입(Enqueue) 단계로 진행한다.

삽입 단계에서는 도착한 메시지를 해당 대기 행렬에 우선 순위를 고려하여 삽입하고 해당 데이터 구조를 갱신하게 된다. 이 단계가 끝나면 최종적으로 다시 감시 단계로 돌아가게 된다.

제거 단계에서는 발견된 관련 메시지들을 전송하는 라우팅(routing) 단계로 진행한다. 이 단계에서는 관련된 메시지들을 명시된 프로토콜에 따라 전송하는 기능을 수행한다. 메시지의 처리는 해당 에이전트에게 송신하는 것뿐만 아니라, 단순 삭제, broadcasting, 다른 DAM으로의 전달(forwarding), 새로운 메시지의 생성 등 프로토콜과 관련된 여러 가지 활동을 포함한다. 이러한 메시지의 라우팅 방식은 메시지의 처리 프로토콜에 따라 다르며, 이는 다음 장에서 설명하게 될 프로토콜 명세(protocol specification)에 명시적으로 정의되어야만 하는 사항이다.

B. 메시지 대기 행렬 (Message Queue)

메시지 대기 행렬은 DAM을 통해 처리되는 메시지들을 지정된 우선 순위(priority)에 따라 입출력 하기 위한 저장 공간이다. 대기 행렬이 처리하는 대상 메시지의 종류에 따라 메시지 대기 행렬과 템플릿 대기 행렬로 구분된다. 전자는 특정 프로토콜에서 데이터 교환을 위해 정의된 메시지 구조에 따라 생성된 일반적인 데이터 내용을 기술하는 데이터 메시지를 저장하며 이는 publish 연산을 통해 생성된다. 후자는 에이전트가 필요로 하는 특정 데이터 메시지에 대한 요구를 템플릿 형태로 표현한 것으로써, subscribe 연산을 통해 생성되는 템플릿 메시지를 저장한다.

C. 연관화 함수 (Association function)

DAM을 통한 메시지의 교환은 데이터 메시지와 템플릿 메시지의 연관화를 통해 이루어진다. 두 메시지가 연관된다는 개념에 대한 정의는 다음과 같다.

Definition [Message Association] - Let m be a data message and t be a template message. Message m and t conforms to a common

message schema S. The message m can be said to be associated with t, only if an association function AF for S should exists, and AF (m, t) return *true*.

위 정의에서 보는 바와 같이 연관화 함수는 같은 메시지 구조를 참조하는 데이터 메시지 m과 템플릿 메시지 t를 인수로 하여 참(true) 혹은 거짓(false)의 값을 반환함으로써 두 메시지의 연관관계를 결정하는 기능을 한다. 즉, 두 메시지의 연관성은 바로 이 함수의 구현 방식에 의해 달라질 수도 있는 것이다. 이러한 연관화 함수는 각 프로토콜에서 정의하는 메시지 타입에 따라서 서로 다른 방식으로 구현되어 제공된다.

3.2. DAMP (Data Associative Messaging Protocols)

DAMP는 DAM을 사용하여 협력적 업무 진행에 참여하는 에이전트들 간의 통신 규약을 정의하는 프로토콜이다. 하나의 프로토콜을 정의하기 위해서는 이를 위해 사용되는 메시지들의 구조(message schema)와 각 구성원들이 담당하는 역할 (participant role), 그리고 이러한 역할별로 협력적 업무에 참여하기 위해 사용하게 되는 인터페이스(interface)가 정의되어야 한다.

A. 메시지 구조 (Message schema)

하나의 프로토콜에서는 여러 가지 서로 다른 종류의 메시지가 사용될 수 있는데, 이러한 개별 메시지의 문법적 구조를 정의한 것이 메시지 구조이다. 메시지 구조에는 해당 프로토콜 에서 사용될 메시지의 타입과 이에 따른 데이터 구조를 XML Schema를 이용하여 기술한다. 또한 필요에 따라 메시지 타입은 기존에 나와 있는 여러 가지 표준 메시지 구조들을 그대로 차용하여 정의할 수도 있다.

DAM를 통한 실질적인 통신은 SOAP (Simple Object Access Protocol)을 통해 이루어지기 때문에, 전달되는 메시지의 구조는 SOAP envelop로 wrapping된다. 즉 하나의 프로토콜에서 정의된 메시지의 타입과 관계없이 SOAP을 위한 메시지 헤더와 바디, 그리고 첨부(attachment) 부분으로 구성된다. 메시지의 바디에는 전달하거나 전달 받을 메시지의 내용이, 헤더에는 메시지를 처리하는 데 필요한 여러 가지 속성들이 기술되며, 여기에 마지막으로 첨부 자료가 추가되어 하나의 전달 가능한 메시지가 정의된다.

B. 역할 (Roles)

역할은 하나의 프로토콜에 참여하게 되는 구성원들의 담당하게 되는 업무를 규정하게 된다. 이는 보통 프로세스 정의 모델에 정의된 조직 구조 모델과 연결되는 것이 일반적이다. 또한 이러한 역할들은 프로세스상의 특정한 업무에 할당되어 있기도 하다.

C. 인터페이스 (Interfaces)

인터페이스는 각 역할별로 프로토콜 내에서 규정되는 행동 방식을 설정한다. 즉, 하나의 프로토콜에는 특정한 역할에 대해서 활용 가능한 인터페이스들을 제공하고, 각 역할에 속하는 에이전트는 이러한 인터페이스를 사용하여 프로토콜 내에서 데이터 교환 및 협력적 업무에 참여하게 된다.

기본적으로 DAMP에는 표 [1]과 같은 기본적인 연산(primitive operation)들을 제공하며, 이러한 연산들을 이용하여 해당 프로토콜의 목적에 부합하는 인터페이스를 정의하고 구현하여 제공할 수 있다. 뿐만 아니라, 정의된 메시지 구조 또한 인터페이스의 정의에 이용됨으로써, 메시지, 역할, 인터페이스의 세 요소는 하나의 프로토콜 내에서 서로 긴밀하게 연관되어 정의되도록 되어있다.

표 [1]. Primitive Operations

Operations	설명
Notify/Listen	이벤트의 통지 및 감시
Request/Service	서비스 요구 및 제공
Call/Return	함수 호출 및 반환
Send/Acknowledge	메시지 전송 및 확인
Receive/ Acknowledge	메시지 수신 및 확인
Read	데이터 읽기
Forward	메시지 전달
Broker	서비스 제공 에이전트 추선

3.3 Example Protocols

앞서 살펴본 메시지 구조, 역할, 그리고 역할에 따른 인터페이스의 정의로 하나의 프로토콜에 대한 명세가 완성된다. 프로세스의 실행과 관련되어 다음과 같은 프로토콜들을 정의하고 활용할 수 있을 것이다.

예를 들어, 업무를 relay 방식으로 진행하기 위한 Relaying protocol을 생각할 수 있다. 이 프로토콜에 참여하는 역할에는 한 명의 initiator와 여러 명의 작업자가 있다. Initiator는 초기 작업명세와 담당자, 그리고 관련 데이터들을 DAM에 생성하고, 이후 진행 과정에 대한 모든 책임을 후임자(작업자)에 위임한다. 생성된 데이터를 DAM으로부터 가져온 첫 후임자는 자신이 해야 할 일부 업무를 처리한 뒤, 적절한 후임 작업자에게 차후 업무를 명세하여 DAM

에 생성한다. 생성된 데이터는 명시된 후임 작업자가 할당 받고, 역시 같은 방식으로 업무 처리 및 전달이 계속 된다. 이러한 Relay과정에서 특정 작업자가 자신이 업무를 수행할 수 없는 상황이 발생한다면, 업무를 취소하거나 위임할 수도 있을 것이다.

이 밖에도, 특정한 이벤트를 발생하고 이를 감시하는 Event listening protocol, 두 참여자의 대화 형식으로 업무를 진행하는 2-way conversation protocol, 서비스 제공자와 서비스 요구자로 구성되어 RPC 형태의 통신을 표현하는 RPC protocol, 그리고 한명의 master와 다수의 slave로 구성되어 업무를 진행하는 master-slave protocol과 같은 다양한 형태의 프로토콜을 정의할 수 있다.

3.4 DAMP의 장점

이와 같이 공유된 데이터 공간을 통한 통신 프로토콜을 비즈니스 프로세스의 자동화에 이용하게 됨으로써 발생하는 장점은 여러 가지가 있지만, 크게 업무 진행 방식의 유연성, 프로세스의 확장성, 그리고 데이터 기반의 업무 진행 등을 들 수 있다.

하나의 프로세스 모델에 대해서도 여러 가지 다양한 형태의 프로토콜을 정의하고 이를 적용함으로써 그 진행 방식을 특정한 형식으로 한정할 필요가 없게 된다. 이는 예외 발생시의 처리 방식이나 업무 진행 방식의 실시간 변경과 같은 요구사항을 만족시키기 위해서 매우 중요한 개념이 된다. 또한, 공유된 데이터 공간에 접근할 수 있는 대상에 대해서 기본적인 제약이 없기 때문에, 프로세스 진행 도중에 새로운 참여자가 업무에 참여하게 될 수도 있고, 새로운 업무나 데이터가 도입될 수도 있다. 마지막으로, 모든 업무의 진행이 데이터의 연관성을 바탕으로 이루어짐에 따라 데이터의 내용에 기반한 업무의 라우팅을 지원함으로써 차후 업무 내용의 분석이 효과적으로 수행될 수 있다..

4. Workflow Enactment through DAMP

본 연구에서 제안하는 DAM을 통한 프로세스 실행 모델의 활용 방안의 하나로 간단한 워크플로우의 실행을 위한 간단 Master-Slave 프로토콜을 명세하고 그 작동 방식에 대하여 설명하도록 하겠다.

4.1 프로토콜 명세(Protocol Specification)

Master-slave 프로토콜은 프로세스의 정의 및 진행을 관할하는 한 명의 Master와 그로부터 업무를 할당 받아 수행하는 다수의 Slave들 간에 진행되는 간단하고 가장 일반적인 워크플로우의 실행 방식이다. 다음의 프로토콜 명세에는 사용되는 메시지 종류와 역할, 그리고 역할에 따른 인터페이스가 개략적으로 기술되어 있다.

```
---------- Master-Slave Protocol Specification ----------

[Message Schemata]
- Process Definition Message PD1
- Job Description Message
- Job Result Message

[Roles]
- Master: MA
- 4 Slaves: SV1, SV2, SV3, SV4

[Interfaces for a Master]
- Enact (Process Definition)
- .....

[Interfaces for Slaves]
- PerformActivity(Session, Job Description);
- ...
-----------------------------------------------------------------
```

이러한 간단한 프로토콜을 이용하여, 예를 들면, Master는 그림 [6]에 나타난 것과 같은 프로세스 모델을 정의된 메시지 구조로 표현한 PD1을 수행하기 위하여, 다음과 같이 구현된 Enact() 인터페이스를 실행할 수 있을 것이다. 한편, PD1에 참여하는 Slave들은 다음과 같이 구현된 PerformActivity() 인터페이스를 사용하여 업무를 할당 받고 그 결과를 Master에게 전달하게 될 것이다.

```
-----------------------------------------------------------------
Enact (Process Definition PD1) {
 JOBTUPLE jt = PD1.FirstJobs;
  while(jt) {
   for(i=0;i<jt.size;i++) Write (sessionID, me, jt(i));
    Read (sessionID, me, ?xxx);
                    // waiting job results.....//
      JOBRESULT jr = xxx;
      jt = PD1.jr.nextJobs;
    }
}

PerformActivity (SessionID sid, Job Description jd) {
 Read (sid, ?who, ?jd);
                                  // blocked until a job
                                    assignment.....//
 fork();
                  //create new tread waiting a new job
                                          assignment//
 JOBTUPLE j = jd;
 JOBRESULT jr = DoJob(j);
                          // blocked until job complete.../
 Write (sid, who, jr);
}
-----------------------------------------------------------------
```

4.2 진행 시나리오

그림 [6] 은 Master-Slave 프로토콜에 따른 워크플로우 PD1의 실행 중 한 단계의 DAM의 모습을 도식화 한 것이다. 현재, Slave SV1과 SV2에 의해 업무 J1, J2가 수행 완료된 상태이며, SV3이 업무 J3를 완료하려고 하는 상황이다. 그림에서, 흰색으로 표시된 번호가 진행 단계를 나타내며, 붉은색 문자는 업무를 기술하는 데이터 메시지, 푸른색 문자는 템플릿 메시지를 나타낸다.

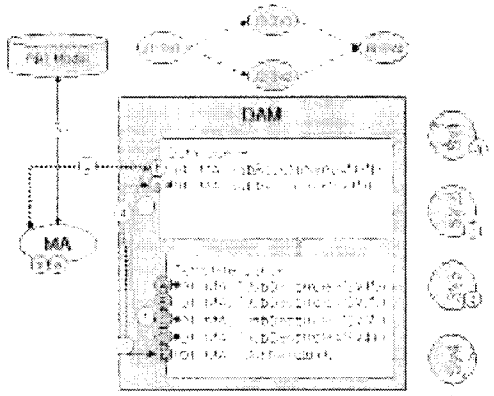

그림 [6] Master-Slave 프로토콜 실행 단계

업무 완료의 과정은 다음과 같은 절차를 따라 진행된다.

① SV3은 업무 J3를 종료하고 작업 결과 데이터를 DAM에 데이터 메시지 f로 기록한다.
② 메시지 f가 Master가 생성한 템플릿 E와 매치된다. 따라서, 작업 결과가 메시지 처리기에 의해 MA로 전송된다.
③ 작업 결과를 전송 받은 MA는 Enact() 인터페이스에 정의된 순서에 따라 PD1 메시지 구조로부터 J3의 다음 작업 명세를 찾아낸다.
④ J3의 다음 작업 명세인 J4를 얻어내고, MA는 그 내용을 담은 데이터 메시지 g를 DAM에 생성한다.
⑤ 단계 2에서 템플릿 메시지 E 가 매치에 의해 잠정적으로 삭제되었기 때문에, MA는 이 메시지를 다시 생성한다.

이렇게 다섯 단계를 거치고 나면, MA는 이제 다시 작업 J4가 완료되기를 기다리는 대기 상태로 돌입하고, 작업 J4는 Slave들이 수행하게 되는 PerformActivity() 인터페이스의 절차에 따라 SV4에게 할당 되게 된다.

이 프로토콜에서는 Master가 임의의 프로세스 모델을 소유하고 그가 구현하는 Enact()인터페이스의 구현 방식에 따라 워크플로우의 진행 방식이 결정된다. 따라서, 이러한 프로토콜에 의해 진행되는 워크플로우의 프로세스 모델은 미리 정해져 있을 필요가 없다. 즉 진행 과정에서 그 상황에 따른 Master의 판단에 따라, 프로세스의 진행 형태가 달라질 수도 있다는 것을 의미한다. 다만, 워크플로우의 진행이 MA에 의해 집중됨으로써, 기존 WFMS의 구동 방식과 유사하며, 많은 트랜잭션이 한곳에 집중되는 단점이 있다.

5. 결론 및 추후 연구과제

본 연구에서는 근래 활발히 대두되고 있는 BPMS(Business Process Management System)의 구현을 위한 프로세스 실행 모델의 구조 및 구현 방안을 제시하였다. 이러한 프로세스 실행 모델은 현재 많이 제안되어 있는 다양한 프로세스 정의 모델과 결합하여 효과적으로 BPMS를 구현할 수 있는 방법론으로 사용될 수 있을 것이다.

뿐만 아니라, 본 연구가 제안하는 코디네이션 메커니즘은 프로세스에 관여하는 참여자들 간에 발생하는 모든 트랜잭션이 기본적으로 하나의 공유된 공간에 기록(logging)되도록 되어 있다. 따라서 이러한 로그 데이터를 분석하면 프로세스의 진행 상황을 모니터링 하거나 여러 가지 다양한 특성 및 성능을 비교 분석할 수 있는 훌륭한 근거 자료를 얻을 수 있다. 이렇게 분석된 자료를 통해 구현된 BPMS는 비단 협력적 E-Business의 자동화된 관리뿐만 아니라, 프로세스 실행 결과의 다양한 분석을 가능하게 함으로써, 진정한 프로세스 리엔지니어링을 가능하게 할 수 있을 것이다.

Acknowledgement
본 연구는 한국과학재단의 특정기초연구과제 (과제번호 R01-2002-000-00155-0)의 지원을 받았음.

6. 참고문헌

Y. Kim et al. (2000), "WW-Flow: Web-Based Workflow Management with Runtime Encapsulation," IEEE Internet Computing, vol. 4, no. 3, pp. 55-64.

WfMC-TC-1003 (1995), The Workflow Reference Model, Workflow Management Coalition, Lighthouse Point, Fla..

D. Gelernter (1985), Generative Communication in Linda, ACM Trans. Programming Languages and Systems, vol. 7, no. 1, pp. 80-112.

G. Cugola, E.D. Nitto, and A. Fuggetta. (2001), "The JEDI Event-Based Infrastructure and Its Application to the

Development of the OPSS WFMS," IEEE Trans. Software Eng., vol. 27, no. 9, 2001, pp. 827-850.

WFMC-TC-1025 (2002), Workflow Process Definition Interface - XML Process Definition Language, Workflow Management Coalition, Lighthouse Point, Fla.

UN/CEFACT and OASIS (2001), ebXML Business Process Specification Schema Version 1.01, www.ebxml.org/specs/ebBPSS.pdf.

RosettaNet (2002), RosettaNet Implementation Framework: Core Specification, March.

A. Arkin et al. (2002), Business Process Modeling Language, www.bpmi.org.

BEA et al. (2002), WSCI 1.0: Web Services Choreography Interface (WSCI) 1.0, http://www.intalio.com/wsci/.

F. Curbera et al. (2002), Business Process Execution Language for Web Services: Version 1.0, http://www.ibm.com/developerworks/library/ws-bpel/.

W3C Note (2001), WSDL 1.1: Web Services Description Language (WSDL) 1.1, World Wide Web Consortium, http://www.w3.org/TR/wsdl.html.