인터넷 서비스 임대를 위한 워크플로우 기반 서비스 중개자 구현기법

이 용 주†

요 약

인터넷 마켓플레이스에서는 사용자를 도와주는 서비스 중개자가 가장 중요한 역할을 수행한다. 최근에 전자상거래나 온라인 쇼핑몰, 그리고 e-마켓플레이스를 위해 많은 지능적 중개자들이 개발되고 있다. 비록 이러한 시스템들은 많은 성공 사례를 보여주고 있지만 인터넷 마켓플레이스에서 요구되는 서비스 결합 및 자동 실행을 지원하는 시스템은 거의 없다. 이러한 관점에서 본 논문에서는 하나의 새로운 워크플로우 기반 서비스 중개자 시스템을 제안한다. 본 연구에서 종속된 서비스들은 워크플로우 프로세스로 표현되고 이 프로세스들은 워크플로우 엔진에 의해 자동 처리된다. 제안된 시스템에서 모든 데이터의 교환은 플랫폼 독립성, 확장성, 개방성, 그리고 상호운용성 구조를 제공하는 XML을 통해 구현된다.

Implementation Techniques of a Workflow-based Service Broker for Service Leasing over the Internet

Yong Ju Lee†

ABSTRACT

A service broker supporting customers plays an important role in internet marketplaces. Recently, several intelligent brokers have been developed for electronic commerce, online shopping mall, and electronic marketplaces. While these systems provide interesting shopping experiences, they fall short in fully exploiting the capabilities, such as the service integration and automatic execution required by internet marketplaces. In this respect, we propose a workflow-based service broker system. Dependent services in our approach are modeled as workflow processes, which are automatically processed by a workflow engine. In the proposed system all data exchanges are realized through XML providing platform independent, extensible, open, and interoperable architecture.

키워드 : 인터넷 마켓플레이스(Internet Marketplaces), 서비스 중개자(Service Broker), 워크플로우(Workflow), XML, 서비스 임대(Service Leasing)

1. 서 론

정보시스템 개발은 지금 중대한 변화의 시기를 맞이하고 있다. 지금까지 하나의 정보시스템을 구축하기 위해서는 데이터베이스, 소프트웨어, 그리고 하드웨어를 모두 구매하여 로컬 컴퓨터에 설치하여 운영하여야만 했었다. 이러한 시스템 구축을 위해서는 많은 노력과 투자가 필요하며, 설치된 시스템에 대한 주기적인 유지 및 보수가 요구되고 소프트웨어 운영을 위한 전문적인 훈련이 필요하다. 하지만 이러한 노력과 투자에 비해 사용자는 설치된 소프트웨어의 모든 기능을 다 사용하는 것이 아니라 그들 중 극히 일부분의 기능만 활용하는 경우가 대부분이므로 낭비적인 요소가 많았다.

최근엔 인터넷을 통해 소프트웨어나 데이터베이스를 빌려주는 임대업이 인기를 끌고 있다[1]. 대표적인 인터넷 임대업으로 응용 서비스 제공자(ASP : Application Service Provider)와 인터넷 데이터베이스 센터(IDC : Internet Database Center) 등이 있다. 인터넷 임대에서 공급자는 인터넷 상에서 데이터나 소프트웨어를 제공하고, 수요자는 필요할 때 이러한 데이터나 소프트웨어를 대여한다. 인터넷 임대 서비스는 인터넷을 통해 제품을 구매하고 배달에 의해 제품을 받는 기존의 전자상거래나 온라인 쇼핑과는 달리, 인터넷에 의해 서비스를 요청하고 인터넷으로부터 서비스를 받는다. 인터넷 임대는 공급자 입장에서는 인터넷상의 수 많은 잠재 고객을 확보할 수 있으며, 수요자 입장에서는 고가의 소프트웨어나 데이터베이스를 구입하지 않아도 되고, 구입 후 유지·관리·보수 등을 걱정할 필요가 없어 그 수요가 급증할 예상이다.

† 정 회 원 : 상주대학교 컴퓨터공학부 교수
논문접수 : 2001년 6월 28일, 심사완료 : 2001년 8월 14일

최근 인터넷 웹 서비스의 기술 발전으로 인해 ASP 및 IDC의 구축이 현실화됨에 따라 많은 시스템들이 앞 다투어 개발되고 있다. 예를 들면, Microsoft, Oracle, SAP와 같은 세계적인 대규모 소프트웨어 기업들은 그들의 차세대 소프트웨어 제품으로써 ASP를 기반으로 한 패키지를 공급할 계획이다[2]. 이러한 추세로 볼 때 분명히 데이터 및 소프트웨어 임대 서비스는 미래 컴퓨터 산업의 한 중요한 패러다임이 될 것이다. 그렇지만 인터넷 서비스 임대는 공급자나 수요자 측면에서 그동안 거의 경험이 없는 새로운 모델이기 때문에 이들이 보편화되기 이전에 해결해야할 많은 경제적, 기술적 문제들이 존재하고 있다. 이들 중 현재 주된 관심은 시스템 내부 기술이나 경제적인 문제보다는 인터넷 상에서 응용 서비스들을 쉽게 배포할 수 있고, 많고 다양한 서비스들 중에서 가장 적합한 서비스를 효율적으로 활용할 수 있는 하나의 인프라스트럭쳐(여기서는 인터넷 마켓플레이스라 한다)의 구축이 요구된다.

일반적으로 인터넷 마켓플레이스(IMP : Internet Market-Places)에서는 데이터와 소프트웨어 서비스를 제공한다[3]. 데이터 서비스는 사용자들에게 인터넷 상에서 데이터의 검색 및 그의 활용을 지원하고, 소프트웨어 서비스는 사용자들에게 소프트웨어나 그 소프트웨어를 수행하기 위한 컴퓨터 하드웨어 사용을 지원한다. IMP에서는 수 많은 수요자와 공급자가 관련되어 있기 때문에 여러 사이트에 분산 저장되어 있는 이질적 서비스들을 쉽게 검색할 수 있어야 하고 필요에 따라 이러한 서비스들을 적절히 결합하여야 할 필요가 있다. 이러한 문제 해결을 위해 IMP에서는 사용자를 도와주는 서비스 중개자(broker)가 가장 중요한 역할을 수행한다[4]. 일반적으로 수요자는 자신이 원하는 것만 기술하고 서비스 중개자는 어떻게 분산된 서비스들이 결합되고 실행되는지를 기술하고 있는 실행 계획을 세운다. 즉, 중개자는 분산된 데이터와 소프트웨어 서비스들을 결합하고 자동적으로 실행되기 위한 계획을 만들어서 사용자가 IMP를 효율적으로 활용할 수 있도록 지원해 준다.

현재 전자상거래나 온라인 쇼핑몰, 그리고 e-마켓플레이스를 위해 많은 지능적 중개자들이 활발히 개발되고 있다[5-7]. 이러한 시스템들은 많은 성공 사례를 보여주고 있지만 IMP에서 요구되는 서비스 결합 및 자동 실행을 지원하는 시스템은 거의 없다. 현재의 시스템들은 대부분 한번에 하나의 질의를 처리하도록 설계되어 있으며 종속적인 서비스 결합 질의를 수행하지 못하고 서비스 자동 실행을 위한 충분한 지원 설비가 없다. 이러한 관점에서 본 논문에서는 워크플로우 기반 서비스 중개자 시스템(workflow-based service broker system)을 제안한다.

워크플로우란 비즈니스 프로세스의 자동화를 의미하며, 워크플로우 관리시스템은 소프트웨어를 이용하여 워크플로우의 수행을 정의, 생성, 관리하는 시스템을 의미한다[8]. 워크플로우는 비즈니스의 목적으로 업무에 참여하는 구성원들 사이에서 이루어지는 문서와 정보 또는 작업의 절차를 정의된 규칙(rule)에 따라 자동화하며, 각 구성원들의 역할(role)과 전체 업무 흐름(routing)을 정의하여 워크플로우를 수행하는데 활용한다. 대부분 워크플로우 관리시스템은 비슷한 구조를 가지고 있는데 일반적으로 프로세스 디자이너, 워크플로우 엔진, 그리고 워크플로우 클라이언트로 구성되어 있다[4]. 본 연구에서는 종속된 서비스들이 워크플로우 프로세스로 표현되고 워크플로우 엔진(workflow engine)에 의해 프로세스들이 자동 처리된다. 특히 IMP에서는 많고 다양한 사용자들을 지원하기 위해 이질적 분산 환경에서의 개방성과 상호운용성을 고려해야 하므로 제안된 워크플로우 기반 서비스 중개자 시스템은 차세대 산업표준 웹 언어인 XML(eXtensible Mark-up Language)[9]을 기반으로 설계되고 구현된다.

IMP 관련 연구로서 MMM[10], SMART[11], Decision-Net[12] 등이 있다. MMM은 개발 초기에 수학분야 IMP로 개발되었으나 현재는 ASP 마켓플레이스로 발전되었다. XML을 이용하여 중개자가 개발되어 있으며, 중개자는 서비스 엔진, 서비스 저장소, 데이터베이스 서버로 구성되어 있다. SMART는 지형공간분야 IMP로서 데이터 제공자(질의 서비스) 및 소프트웨어 제공자(함수 서비스) 기능으로 중개자가 구성되어 있다. 이들 기능들은 자체 RSL(Request Specification Language) 작성에 의해 질의되고 실행된다. Decision-Net은 의사결정분야를 위한 IMP로서 에이전트에 의한 자료수집, 자료변환, 자료입력, 그리고 서비스 실행 작업을 수행한다. 이들 IMP 시스템들의 핵심 부분은 중개자이지만 본 논문에서 제안하는 서비스 결합 및 자동실행에 관한 기능은 거의 없다. 특히, 이러한 기능들의 중요성은 강조하고 있으나 본 연구와 같은 워크플로우 기반 중개자 기능은 없다.

본 논문의 구성은 다음과 같다. 2장에서는 인터넷 마켓플레이스 구조를 설명하고, 3장에서는 서비스 중개자에 대해 자세히 기술하였다. 특히, 실행계획 및 이의 분석모델을 설명하였다. 4장에서 시스템을 구현하고 5장에서 결론을 내렸다.

2. 인터넷 마켓플레이스의 구조

일반적으로 IMP는 특정 응용분야에 대한 수요자 및 서비스 공급자로 구성된다. 서비스 공급자는 두 분류 : 데이터 서비스 제공자(DSP : Data Service Provider)와 응용 서비스 제공자(ASP : Application Service Provider)로 나눠진다. 수요자와 공급자는 통신 네트워크(인터넷)를 통해 서로 연결된다. 본 장에서는 먼저 하나의 전형적인 IMP 응용 예를 통해 그 유용성을 살펴본다.

[예 1] 한 연구가는 주어진 지점에서 가장 가까운 객체를 찾아내는 Nearest Neighbor 알고리즘을 개발했다(그림

1). 이는 지형공간분야에서 자주 사용되는 알고리즘이므로 많은 사람들과 공동 사용되기를 원한다. 어떤 방법이 가장 적합할까? 기존의 가장 일반적인 방법으로는 이 알고리즘을 FTP 사이트에 등록하여 관심 있는 사람들이 다운로드 받을 수 있도록 할 수 있다. 이 방법은 사용자가 이를 실행하기 위해서는 자신의 로컬 컴퓨터에 적합한 실행 환경(예, 컴파일러 등)이 갖추어져 있어야만 하고, 컴파일 시 PointToPointDistance 함수가 준비되어 있어야만 한다. 이는 상황에 따라 많은 시간과 노력이 낭비될 수 있다. 만일 알고리즘이 복잡하다면 이 상황은 더욱 심각해진다.

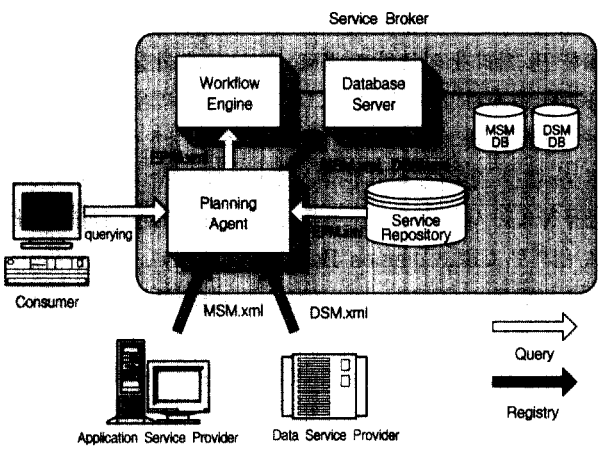
반면에, IMP에서는 공급자가 쉽게 소프트웨어(알고리즘, 스크립트, 매크로 등)을 배포하고, 사용자가 편리하게 활용할 수 있는 하나의 새로운 프레임워크를 제공하고 있다. 공급자를 위해 IMP는 권한이 있는 사용자만 접근되도록 통제하고 소프트웨어 간 상호운용 및 원격 서비스가 처리되도록 지원한다. 또한 사용자는 소프트웨어에 대한 특별한 지식 없이 단지 웹 브라우저의 사용법만 알면 원하는 업무를 효율적으로 수행할 수 있도록 지원해 준다. □

```
Algorithm 1 : NearestNeighbor (P0, S)
// input : a point P0 and a dataset S
// output : find the closest object from a given point
  for each Pi ∈ S
    distance Di = PointToPointDistance (P0, Pi)
    // distance = √( (p0,x − p1,x)² + (p0,y − p1,y)² )
    nearest = MIN(Di)
  end-for
  return (nearest)
End Algorithm1
```

(그림 1) Nearest Neighbor 알고리즘

IMP의 전체적인 구조는 (그림 2)와 같이 서비스 공급자(provider)와 수요자(consumer), 그리고 서비스 중개자(service broker)로 구성되어 있다. 서비스 중개자는 계획 에이전트(planning agent), 서비스 저장소(service repository), 워크플로우 엔진(workflow engine), 데이타베이스 서버(database server)로 이루어져 있으며, 중개자는 IMP의 미들웨어(middleware) 역할을 수행한다. 이에 대한 자세한 설명은 다음과 같다.

- 서비스 공급자(ASP 또는 DSP) : 제공되는 서비스 정보를 저장하고 있다. 사용자들에게 서비스 질의 및 실행을 지원한다.
- 수요자 : 웹 브라우저를 통해 계획 에이전트에 접속한다.
- 계획 에이전트 : 사용자의 요청을 받아들여서 관련 정보를 검색·조작·통합한다.
- 서비스 저장소 : 서비스 공급자로부터 등록되는 메타데이터(metadata)를 저장하고 있다. 또한, 계획 에이전트에서 사용되는 실행 계획 메타데이터도 저장되어 있다.
- 워크플로우 엔진 : 프로세스 자동화를 위해 워크플로우의 수행을 정의·생성·관리하는 시스템이다.
- 데이터베이스 서버 : 데이터와 소프트웨어 메타데이터를 데이터베이스에 저장한다. 서버에서 XML 문서들이 관계형 데이터베이스 테이블로 자동 변환된다.



(그림 2) 인터넷 마켓플레이스의 구조

IMP에서 공급자는 메타데이터를 사용하여 서비스를 등록할 수 있어야 하고 수요자는 이 메타데이터를 이용하여 자원을 검색할 수 있어야 한다. (그림 2)에서는 세 종류의 메타객체(즉, DSM, MSM, EPM)가 있는데, 1) DSM(Data Service MetaObject)은 데이터 서비스를 표현하고, 2) MSM(Method Service MetaObject)은 소프트웨어 서비스를 표현하며, 그리고 3) EPM(Execution Plan MetaObject)은 실행 계획을 표현하고 기술한다. 각각의 구성 및 역할은 다음과 같다.

- DSM : DSM은 온라인 서비스 데이터 또는 소프트웨어 입력용 데이터셋(dataset)을 표현하는 메타객체이다. DSM은 서비스 공급자, 사용권한, 데이터 포맷, 타입, 위치, 크기, 그리고 분류 카테고리 등을 기술하고 있다. 특히, 사용권한은 "누가 어떤 방법으로 데이터를 액세스할 수 있는가?"를 표현하고 있다. DSM에는 파일과 데이터베이스를 포함할 수 있다. 파일을 위해서는 압축여부 및 포맷 등을 제공하고, 데이터베이스를 위해서는 데이터베이스 접근방법(예, ODBC, OLE/DB) 등을 제공한다.
- MSM : MSM은 IMP에서 제공되는 소프트웨어 서비스를 표현하는 메타객체이다. MSM은 서비스 공급자, 위치, 입출력 매개변수, 서비스 수행 환경, 그리고 분류 카테고리 등을 기술하고 있다. MSM은 하나의 독립적인 메소드에 관한 모든 정보를 캡슐화(encapsulation)하고 있으므로 복

수의 연속적인 메소드 실행 계획을 수립하기 위한 기반이 될 수 있다.

- EPM : EPM은 서비스 실행 계획을 기술하고 있는 메타객체이다. EPM은 검색 키워드와 함께 메소드 실행 순서(워크플로우)를 기술하고 있다. 즉, EPM은 IMP에서 어떻게 데이터와 메소드가 검색되고 실행되는지 그 순서를 묘사하고 있다.

이들 메타객체들은 차세대 웹 표준 언어인 XML로 작성되어 사용자가 서비스를 검색하거나 실행할 때 사용되어진다. XML을 사용한 이유는 시스템 간에 플랫폼 독립적인 데이타 교환 및 처리가 가능하고, XML은 시스템 확장성 및 상호운용성을 보장하기 때문이다. 그렇지만 각 조직에서 자기 자신만의 XML 태그를 사용한다면 시스템간 상호운용성은 보장될 수 없을 것이다. 이러한 관점에서 많은 비즈니스 도메인에 공통적으로 사용되는 요소들을 DTD(Data Type Definition)로 정의한 CBL(Common Business Library), cXML(Commerce XML), IOTP(Internet Open Trading Protocol), OAGIS (Open Applications Group Integration Specification), 그리고 BizTalk 프레임워크 등이 개발되었다[13]. 그러나 이러한 프레임워크들 조차도 아직까지는 세계적으로 통용될 수 있는 상황은 아니고 독자적으로 표준화를 진행 중에 있다. 따라서 본 연구에서는 자체 XML DTD를 개발하였고 향후 CBL 등이 활용 가능하면 이를 대체할 수 있도록 하였다.

DTD는 XML 문서에 포함될 수 있는 각각의 요소가 무엇인지를 정의하며 문서의 전체적인 구조를 나타내고 있다. 본 논문에서 사용하는 IMP 메타객체들에 대한 XML DTD는 부록에 기술하고 있다. 하나의 XML 문서는 IMP 메타객체의 하나의 인스턴스(instance)를 표현하며, (그림 3)과 (그림 4)는 부록에 정의된 DTD 규칙에 따르는 DSM과 MSM 인스턴스를 나타내고 있다. EPM 인스턴스는 다음 장에 설명하고 있다.

```
<dsm id = "dsm200162501">
<name> point_dataset </name>
<general>
  <person type = "provider" lastname = "Brian" firstname = "Richard" middle = "">
    <address> … </address>
    <email> richard@korea.ac.kr </email>
  </person>
  <creation year = "2001" month = "6" day = "25"> </creation>
  <version> 1.0 </version>
  <rights> <owner> 7 </owner> <group> 5 </group> <other> 5 </other> </rights>
</general>
<access>
  <file decompression = "none" format = "ascii" protocol = "http">
    <addr> http : //www.korea.ac.kr/~richard/dataset </addr>
    <type> <list> integer </list> </type>
  </file>
 </access>
<domain>
  <category name = "Geometric dataset">
    <attribute name = "data">
    <value> point_dataset </value>
    </attribute>
  </category>
  <description> Point geometric dataset </description>
  <keyword> point </keyword>
</domain>
</dsm>
```

(그림 3) DSM 인스턴스

```
<msm id = "msm200162501">
<name> nearest_neighbor </name>
<general>
  <person type = "provider" lastname = "Brian" firstname = "Richard" middle = " ">
    <address> … </address>
    <email> richard@korea.ac.kr </email>
  </person>
  <creation year = "2001" month = "6" day = "25"> </creation>
  <version> 1.0 </version>
</general>
<access src = "http : //www.korea.ac.kr/~richard/nearest_neighbor.C"
  type = "source" language = "C++"> </access>
<interface>
  <call> nearest_neighbor (P, S) </call>
  <input>
    <parameter name = "P"> <type> <array rows = "2"> <type>
    real </type>
      </array> </type>
    <description> A point P (x, y) </description> </parameter>
    <parameter name = "S"> <type> <file decompression = "none"
    format = "ascii"
      protocol = "http"> <addr> http : //www.korea.ac.kr/~richard
      /dataset </addr>
      </file> </type> <description> Spatial dataset </description>
      </parameter>
  </input>
  <output>
    <parameter name = "P"> <type> <array rows = "2"> <type>
    real </type>
      </array> </type>
    <description> The closest point from a given point </description>
    </parameter>
  </output>
</interface>
<domain>
  <category name = "Geometric Algorithms">
    <attribute name = "method">
    <value> nearest_neighbor_query </value>
    </attribute>
  </category>
  <description> Nearest neighbor query </description>
  <keyword> nearest </keyword> <keyword> neighbor
  </keyword>
```

```
    <paper href = "http : //info.korea.ac.kr/~richard/distance.ps">
    </paper>
</domain>
</msm>
```

(그림 4) MSM 인스턴스

3. 서비스 중개자

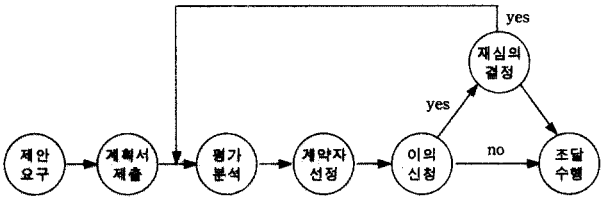
IMP의 여러 기능들 중에서 데이터와 소프트웨어 서비스들을 결합하고 실행하기 위해 수행순서(계획)를 만들어서 사용자가 사용하기 쉽도록 지원해 주는 서비스 중개자가 가장 중요한 역할을 수행한다. 서비스 중개자는 수요자에게 서비스들을 즉시 실행할 수 있는 프로그램(계획)을 제공하므로 검색기와는 그 기능이 다르다. 검색기는 어떤 요구사항을 만족하는 서비스의 이름과 간단한 설명만 제공한다. 실행 계획을 생성하기 위해서는 EPM 메타객체가 이용되는데 이 정보는 서비스 저장소에 사전에 준비되어 있는 워크플로우 템플릿을 기반으로 사용자가 서비스를 검색할 때 자동적으로 만들어진다.

3.1 실행 계획

계획은 포함하고 있는 서비스의 개수에 따라 간단한 계획과 복잡한 계획으로 분류할 수 있다. 간단한 계획은 단지 한 개의 서비스만 호출한다. 반면에 복잡한 계획은 서비스들의 연속적인 호출을 요구한다. 복잡한 계획의 성능은 어떻게 수많은 서비스들을 효율적으로 결합할 수 있는가에 달려있다. 예를 들면, 한 고객은 표준 데이타 형식(예, SDTS 형식)으로 작성된 서울 지역의 전자 지도를 요구할 수 있다. 그렇지만 이 지도를 제공할 수 있는 공급자는 현재 지도 작성을 위해 가장 많이 사용하고 있는 ArcInfo 시스템용 지도만 제공할 수 있다고 가정한다. 그러면 계획 에이전트는 ArcInfo 데이타를 SDTS 데이타로 변환할 수 있는 소프트웨어를 검색하여 두개의 서비스로 구성된 하나의 계획을 만들 수 있다. 첫 번째는 ArcInfo용 데이타를 호출하고 두 번째는 데이타 변환 소프트웨어를 호출하는 계획을 만든다.

IMP에서 실행 계획의 작성은 분산된 서비스들의 결합 및 프로세스 자동 실행을 지원해야 하기 때문에 기존의 웹 검색 알고리즘보다 훨씬 복잡하고 구현하기 어렵다. 기존의 검색기들은 대부분 한번에 하나의 질의를 처리하도록 설계되어 있으며 종속적인 서비스 결합 및 질의를 수행하지 못하고, 프로세스 자동화를 위한 충분한 지원 설비가 없다. 이에 반해 본 연구에서는 실행 계획을 워크플로우 프로세스로 표현함으로써 종속적인 서비스 프로세스들이 결합될 수 있으며 이 프로세스들은 워크플로우 엔진에 의해 자동 처리된다.

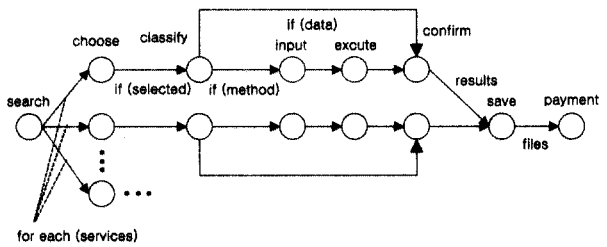
전통적인 워크플로우 프로세스 구조에는 순차, 병행(AND), 선택(OR), 반복(LOOP) 등이 있다[14]. (그림 5)는 조직에서 물품을 조달하는 워크플로우의 예로 7개의 작업들로 구성되어 있다. 이러한 워크플로우의 정의, 실행, 관리는 워크플로우 엔진에서 수행된다.



(그림 5) 물품조달 워크플로우

다음 예는 전통적인 워크플로우 개념을 응용하여 하나의 실행 계획을 워크플로우 프로세스로 묘사한 것이다.

[예 2] (그림 6)은 IMP에서 일어날 수 있는 실행 계획을 워크플로우 프로세스로 도형화 한 것이다. 워크플로우는 연속적으로 호출되는 서비스들로 구성되어진다.



(그림 6) 실행 계획 워크플로우

본 논문에서는 이질적 분산 환경을 효과적으로 지원하기 위해 워크플로우 프로세스를 XML로 기술하였다. 워크플로우 프로세스 XML 문서는 태스크(task)와 블록(block)의 집합으로 구성된다. 태스크는 입력과 출력 매개변수를 가지고 있으며, 블록은 3개의 형태 즉, 순차(serial), 반복(for each), 조건(condition)을 가지고 있다. 순차 블록에서는 단위 업무(activity)들이 하나씩 순차적으로 수행되고, 반복 블록은 리스트의 각 멤버들을 반복적으로 수행한다. 그리고 조건 블록은 조건 결과에 따라 단위 업무가 수행된다. 종속적인 서비스 결합을 지원하기 위해 하나의 XML 문서에는 복수의 워크플로우 프로세스가 정의될 수 있다. (그림 7)은 (그림 6)의 워크플로우를 XML 문서로 표현한 것이다. □

(그림 7)에 작성된 워크플로우 XML 문서는 EPM 인스턴스 작성을 위한 템플릿(template)으로 사용된다. 즉, EPM 인스턴스는 이 템플릿에 맞추어 작성된다. 워크플로우 템플릿을 위한 DTD 파일은 부록 3에 나와 있으며, 사용자는 이 DTD에 따르는 다양한 템플릿들을 구성할 수 있다. EPM 인스턴스를 파싱하여 관련 서비스를 실행시키기 위하여 본 연구에서는 W3C(World Wide Web Consortium)에서 정한 DOM

```
<epms>
<epm id = "template01">
<name> execution plan template </name>
<service> data or method </service>
<keyword> keyword </keyword>

<serial-block>
 <task name = "search">
   <input> <parameter name = "keyword"> <type> string </type>
    </parameter> </input>
   <output> <parameter name = "services"> <type> <array rows
   = "n"> <type> string
          </type> </array> </type> </parameter> </output>
 </task>
 <for-each-block>
     <task name = "choose">
      <input> <parameter name = "services"> <type> <array rows
       = "i"> <type> string
          </type> </array> </type> </parameter> </input>
      <output> <parameter name = "selected"> <type> integer
          </type> </parameter> </output>
     </task>
     <condition-block>
      <condition> <lhs-expr> selected </lhs-expr> <comp-opr
      type = "EQUAL">
          </comp-opr> <rhs-expr> <int> 1 </int>  </rhs-expr>
          </condition>
      <task name = "classify"> ··· </task>
     </condition-block>
     <condition-block>
      <condition> ··· </condition>
      <task name = "execute"> ··· </task>
     </condition-block>
 </for-each-block>
 <task name = "save">
   <input> <parameter name = "results"> <type> <list> string
          </list> </type> </parameter> </input>
   <output> <parameter name = "files"> <type> <list> string
          </list> </type> </parameter> </output>
 </task>
 <task name = "payment"> ··· </task>

</serial-block>
</epm>
</epms>
```

(그림 7) 워크플로우 XML 문서

(Document Object Model) 인터페이스[15]를 사용하였다. DOM을 이용하여 (그림 7)의 XML 문서를 서비스 실행 알고리즘으로 변환하면 (그림 8)과 같다.

먼저 사용자가 키워드로 질의하면 관련된 서비스 리스트가 나열된다. 리스트 중 사용자가 선택한 항목에 대해 이 서비스가 메소드면 입력 데이터가 수집되고 메소드가 실행된 후 그 결과를 확인한다. 만일 데이터 서비스라면 바로 결과를 확인할 수 있다. 그 다음, 산출된 결과를 저장하고 사용량에 따라 요금이 부가된다. 이 알고리즘에 의해 어떻게 서비스들이 IMP에서 수행되는지 간략하게 알 수 있다. IMP에서 워크플로우 프로세스들은 워크플로우 엔진에 의해 실행되며, 이의 수행 과정은 (그림 2)에서 진한 화살표로 표시하였다.

IMP에서는 수요자와 공급자간에 종속적인 서비스 결합이 필요한 경우가 자주 있다. 다음 예는 종속적인 서비스 결합을 지원하기 위한 하나의 시나리오를 묘사하고 있다.

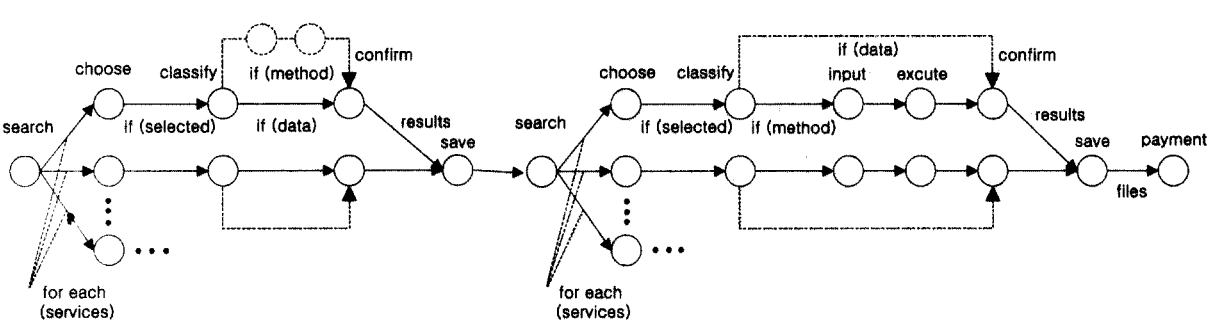
```
Algorithm 2 : WorkFlow

for each plan ∈ epms
 services = search(keyword) ;
 for each service[i] ∈ services
   if choose(service[i]) = selected
     if classify(service[i]) = method
       inputData = input(service[i], parameters) ;
       outputData = execute(inputData) ;
     end-if
   results = confirm(outputData) ;
   end-if
 end-for
 files = save(results)
 payment(usageInfo) ;
end-for

End Algorithm 2
```

(그림 8) 계획 실행 알고리즘

[예 3] 현재 위치에서 가장 가까운 병원을 찾는 한 환자가 있다. 이 사용자는 웹 브라우저에서 조회하기를 원하는 데이터 및 메소드 키워드를 입력한다. 찾고자 하는 데이터와 메소드는 주변의 위치정보와 최근접(nearest neighbor) 질의 서비스가 될 수 있다. 질의 후 서비스 중개자의 계획 에이전트는 서비스 저장소에 이미 저장되어 있는 워크플로우 템플릿을 사용하여 XML로



(그림 9) 종속적 서비스 결합을 위한 워크플로우

작성된 EPM 인스턴스를 만든다. 참고로, (그림 9)는 워크플로우 템플릿을 적용한 최종 워크플로우 프로세스를 보여주고 있다. □

3.2 실행계획 분석모델

실행계획의 효과를 분석하기 위해 본 연구에서는 경제학에서 널리 활용되고 있는 비용/효과 모델[16, 17]을 본 시스템에 적용시켰다. 우리 서비스 중개자는 메타객체들을 이용하여 간단한 키워드 검색으로부터 복잡한 서비스 결합 및 자동 실행까지 다양한 서비스들을 제공하고 있다. 그렇지만 효과분석을 위해 서비스 중개자는 간단한 서비스(low quality service)와 복잡한 서비스(high quality service) 두 가지만 제공한다고 가정한다. 간단한 서비스는 EPM없이 DSM과 MSM 메타객체만 이용하여 적합한 공급자를 검색하고 수요자와 공급자를 연결시켜주는 서비스를 말한다. 복잡한 서비스는 EPM 메타객체를 사용하여 서비스 결합 및 실행을 위한 실행 계획을 작성하여 워크플로우 엔진에 의해 종속된 서비스들이 자동 실행되는 서비스를 말한다.[1)]

간단한 서비스(L)와 복잡한 서비스(H)의 질(quality)을 각각 Q_L과 Q_H라하고, 수요자가 제공받는 서비스 수준에 따라 중개자에게 지불하는 요금(fee)을 각각 P_L과 P_H라 한다. 수요자가 공급자에게 지불하는 요금은 중개자 분석 모델에서는 고려하지 않는다. 제공되는 서비스에 대한 수요자 만족도를 θ라 하면 수요자의 손익(G)은 다음과 같이 표현된다.

$$G_L = \theta \cdot Q_L - P_L \quad (1)$$

$$G_H = \theta \cdot Q_H - P_H \quad (2)$$

여기서 θ의 값은 간격 [0, 1] 사이에 존재한다.

서비스 L과 H를 선택한 수요자의 비율을 각각 N_L과 N_H라하고, 중개자를 활용하는 공급자의 비율을 M, 공급자가 중개자에게 지불하는 요금을 δ라고 하면 중개자의 이익 함수(B)는

$$B = P_L \cdot N_L + P_H \cdot N_H + \delta \cdot M \quad (3)$$

로 표현된다.

(그림 10)은 수요자의 이익과 비용을 그래프로 표현한 것이다. 서비스 L의 손익 분기점 θ_L과 서비스 L 및 H가 이익이 같은 지점 θ_E는

$$\theta_L = \frac{P_L}{Q_L}, \quad \theta_E = \frac{P_L - P_H}{Q_L - Q_H} \quad (4)$$

로 표현된다. 수요자는 이익이 되는 경우에만 서비스를 활용하고 이익이 더 많이 되는 서비스를 선택할 것이다. 따라서 서비스 L과 H를 선택한 수요자 비율은 각각

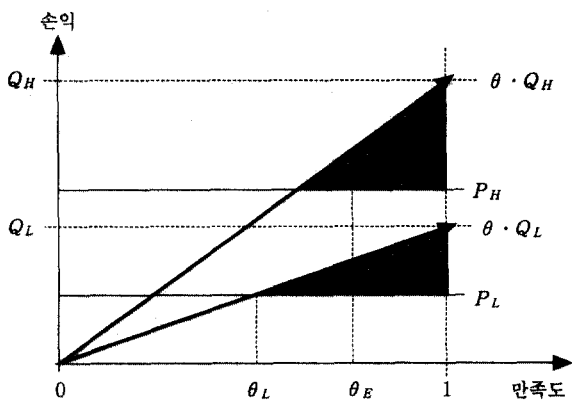
$$N_L = \theta_E - \theta_L, \; N_H = 1 - \theta_E \quad (5)$$

가 된다. 또한, 공급자 이익 함수가 [0, 1] 사이에 균등분포 한다고 가정할 때

$$M = 1 - \delta \quad (6)$$

가 된다. 식 (5), (6)을 식 (3)에 대입하면 B는 다음과 같은 비용 함수로 표현된다.

$$B = P_L \cdot (\theta_E - \theta_L) + P_H \cdot (1 - \theta_E) + \delta \cdot (1 - \delta) \quad (7)$$



(그림 10) 수요자의 이익과 비용

최적 가격을 결정하기 위해 각 요금에 대한 1차 미분을 수행하면

$$\frac{\partial B}{\partial P_L} = 2(\theta_E - \theta_L), \quad \frac{\partial B}{\partial P_H} = 1 - 2\theta_E,$$

$$\frac{\partial B}{\partial \delta} = 1 - 2\delta$$

와 같이 된다. 따라서 최적 손익 분기점은

$$\theta_L = \theta_E = \frac{1}{2}$$

최적 가격은

$$P_L = \frac{Q_L}{2}, \quad P_H = \frac{Q_H}{2}, \quad \delta = \frac{1}{2}$$

서비스를 선택한 수요자 및 공급자 비율은

$$N_L = 0, \quad N_H = \frac{1}{2}, \quad M = \frac{1}{2}$$

이 가격에서 중개자의 이익은

$$B = \frac{Q_H}{4} + \frac{1}{4}$$

1) 본 논문에서 제안된 내용은 복잡한 서비스에 속한다.

가 된다. 첫 번째는 서비스 H를 선택한 수요자로부터 얻은 이익이고 두 번째는 공급자로부터 얻은 이익이다.

결과를 분석하면, 최적 가격에서 서비스 L은 어느 누구도 선택하지 않지만 서비스 H는 전체 수요자의 50%가 선택한다. 즉, 이론적으로는 IMP에서 간단한 서비스는 제공할 필요가 없고 다만 복잡한 서비스(실행계획에 의한 서비스)만 전략적으로 제공하면 된다.

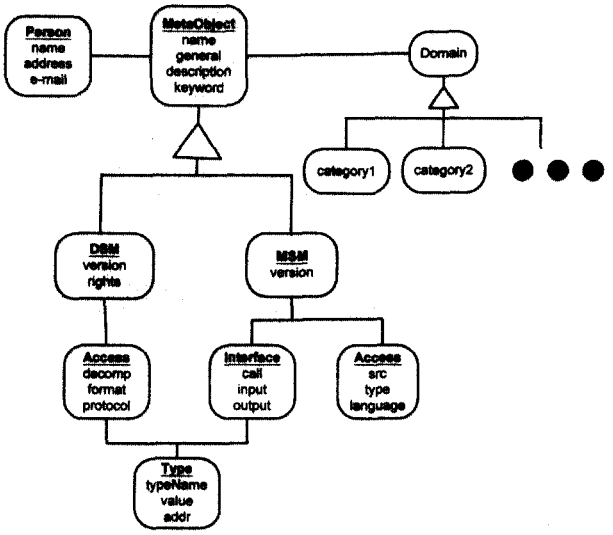
4. 시스템 구현

본 연구에서는 웹(Web) 프로그래밍을 위해 Microsoft의 ASP+(Active Server Pages plus)를 사용하였고, 워크플로우 엔진을 구현하기 위해 COM+(Component Object Model plus)를 사용하였다[18]. COM+컴포넌트 서비스는 다층(multi-tier) 구조 분산시스템 구현을 위한 미들티어(middle-tier) 트랜잭션 관리 메커니즘 역할을 수행한다. ASP+와 컴포넌트 사이의 자료 교환은 XML을 통해 이루어진다.

4.1 메타 데이터베이스 구축

IMP에서 공급자들은 그들의 서비스를 쉽게 등록할 수 있고, 쉽게 분류·결합할 수 있는 간단한 인터페이스를 기대한다. 따라서 등록 과정은 웹 브라우저의 폼 형태로 입력되며 공급자는 미리 준비된 분류 항목을 조사하여 등록 내용이 속할 카테고리를 선택한다. 서비스 제공자는 지역적으로 분산되어 있으므로 공급자는 자료를 전송하기 전에 먼저 계획 에이전트에게 자신의 컴퓨터 위치(URL)를 알려준다. 공급자와 에이전트간에는 XML과 HTTP 프로토콜을 이용하여 통신이 이루어진다. 이러한 표준화된 접근 방법은 각 계층의 컴퓨터가 어떤 소프트웨어, 운영체제, 그리고 어떤 데이타베이스 시스템을 사용하던 간에 상관없이 작동될 수 있다. 2장에 있는 (그림 3)과 (그림 4)는 전송되는 XML 문서의 예를 보여주고 있다. 계획 에이전트에서는 공급자로부터 온 XML 문서를 데이터베이스 서버에게 전달하며, 데이터베이스 서버에서는 XML을 파서(parse)하여 SQL(Structured Query Language) 입력문을 생성하고 이를 실행하여 서비스 데이터베이스를 구축한다. 현재 SQL문 실행을 위해 관계형 DBMS인 SQL Server 2000을 사용하고 데이터베이스의 검색·삽입·수정을 위하여 ADO(ActiveX Data Object)를 사용한다.

(그림 11)은 메타 데이터베이스 스키마를 UML(Unified Modeling Language) 클래스 다이어그램[19]을 사용하여 모델링한 것이다. 메타객체(MetaObject)는 각각 데이터와 메소드를 표현하고 있는 DSM과 MSM으로부터 구축된다. 메타객체는 서비스 공급자와 키워드를 표현하고 있고, 여러 도메인에 의해 분류된다. 메타객체와 DSM, MSM은 일반화(generalization) 관계를 형성한다. DSM은 엑세스(Access)로, MSM은 인터페이스와 엑세스로 구성될 수 있으며, DSM 엑세스와 MSM 인터페이스는 타입과 관계를 가진다. UML 클래스 다이어그램으로 기술한 개념적 스키마는 관계 DBMS에 저장되기 위해서는 논리적 스키마로 변환되어야 한다. 클래스들은 직접 관계 테이블로 사상되지만 일반화 관계를 나타내는 부모와 자식 클래스는 동일한 id를 갖도록 구성된다.

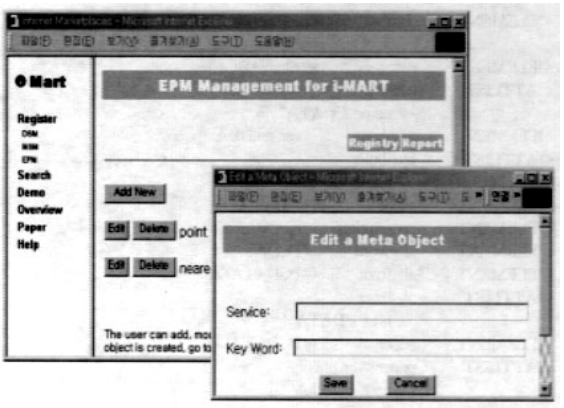


(그림 11) 메타 데이터베이스 스키마

4.2 서비스 질의

IMP에서 수요자는 EPM 메타객체를 이용하여 종속적 서비스 결합 및 질의를 쉽게 조작할 수 있어야 한다. 먼저 사용자는 웹 브라우저를 사용하여 EPM 메타객체를 추가·변경·삭제할 수 있다. 즉, (그림 12)의 왼쪽 등록항목에서 EPM을 선택하면 EPM 등록화면이 나타난다. 등록화면에는 추가, 변경, 삭제 버튼이 있다. 사용자가 하나의 워크플로우를 추가하거나 변경할 필요가 있을 때 추가나 변경 버튼을 클릭하며, 하나의 워크플로우를 삭제하고 싶을 때 삭제 버튼을 클릭한다. 추가 버튼과 변경 버튼을 클릭하면 입력 및 편집 폼을 가진 새로운 브라우저 창을 연다. 새창에서 "서비스"와 "키워드"를 입력하면 추가 시에는 워크플로우 템플릿이 포함된 하나의 새로운 워크플로우 객체가 생성되고, 변경 시에는 기존의 객체에서 서비스와 키워드가 변경된다. 복수의 종속적인 워크플로우를 생성하기 위해서는 이런 작업을 반복함으로써 쉽게 작성할 수 있다. 삭제 버튼은 XML 문서에서 관련된 워크플로우 객체를 지우는 일을 한다. 보고서 화면에서는 작성된 XML 문서를 보여준다.

수요자가 작성한 EPM 메타객체를 기반으로 질의를 요청하면 계획 에이전트는 Microsoft의 MSXML DOM 인터페이스를 사용하여 XML 문서를 파싱하고, (그림 8)과 같은 계획 실행 알고리즘을 작성한다. 이때 계획 에이전트는 워크플



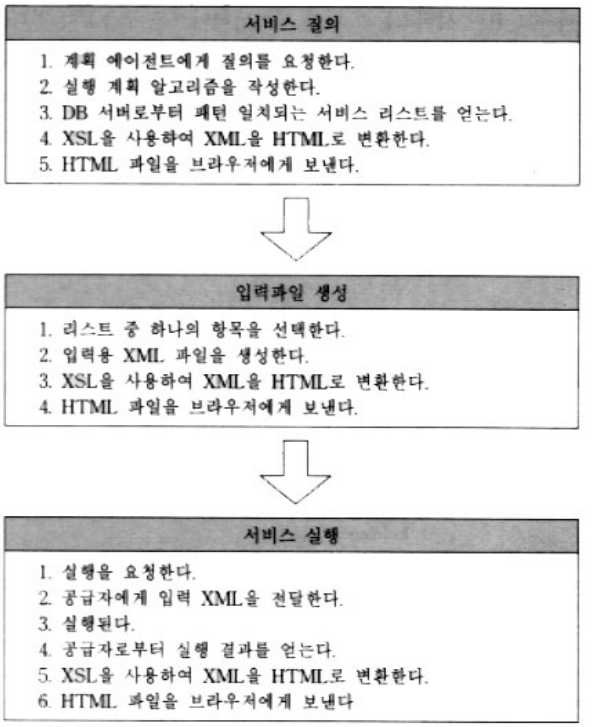
(그림 12) EPM 등록 화면

로우 엔진과 상호 작용하여 워크플로우 프로세스들을 수행한다. 구체적으로 설명하면, 사용자는 웹 브라우저에서 찾고자 하는 키워드를 입력한 후 계획 에이전트에게 질의를 요청한다. 계획 에이전트는 데이터베이스 서버를 통해 패턴이 일치하는 서비스들을 나열한다. 이 서비스 리스트는 XML 문서로 작성된다. XML로 만들어진 리스트는 XSL(eXtensible Stylesheet Language) 스타일시트를 사용하여 HTML로 변환되어 웹 브라우저에게 보낸다.

사용자가 리스트에 있는 하나의 아이템을 선택하면 계획 에이전트는 이 요청 서비스가 메소드인지 데이터인지 판단한다. 만일 메소드이면 MSM 메타객체를 이용하여 입력 데이타를 수집한다. 입력 완료 후 에이전트는 메타객체 정보를 이용하여 데이타 타입을 변환(예, integer 또는 real로 변환)하고 데이타를 결합하여 입력용 XML 파일을 생성한다. 이것도 XSL 언어를 사용하여 HTML 파일로 변환된 후 브라우저에 디스플레이 된다.

사용자는 자세한 사항을 검토한 후에 이 내용의 실행을 요청한다. 이때 계획 에이전트는 공급자 시스템에게 입력 XML 파일을 전달하고 관련 작업이 실행되도록 요청한다. 공급자 시스템은 결과가 포함된 XML 파일을 생성하고 XSL 스타일시트를 사용하여 XML을 다시 HTML 파일로 변환한 후 그 결과를 사용자에게 보낸다. 이러한 프로그램 처리 과정은 (그림 13)과 같으며 각 단계별로 클라이언트가 관련 메쏘드들을 계획 에이전트로부터 호출한다.

(그림 13)에서 중요한 구현 기법은 XSL 스타일시트와 컴포넌트 객체의 사용이다. XSL은 XML 문서의 외형을 만들어 내는 매우 융통성 있는 도구로써, XML 파일과 XSL 파일이 결합하여 HTML 파일을 만들 수 있으며 데이타를 정렬하거나 해당되는 내용을 찾기 위해 패턴 매칭을 할 수 있다. COM+ 컴포넌트 객체의 사용은 ASP+ 프로그램에서 서버 엔진의 자세한 구현 사항을 숨길 수 있으므로 프로그램 작성을 간단히 하고 이의 해독력을 향상시킨다.



(그림 13) 프로그램 처리과정

본 연구에서는 데이타 교환을 위해 XML을 사용함으로써 플랫폼 독립성을 지원하고 서버와 클라이언트 쪽에 유연성을 향상시킨다. 클라이언트에서는 서버가 어떤 데이타베이스 시스템이나 파일 시스템을 사용하더라도 자유스럽게 응용 프로그램을 구성할 수 있다. 서버쪽에서도 클라이언트가 어떤 프로그램을 수정하더라도 서버 프로그램을 수정할 필요가 없으므로 시스템 독립성을 유지할 수 있다.

5. 결 론

본 논문에서는 인터넷 서비스 임대를 위한 워크플로우 기반 서비스 중개자 시스템을 제안하였다. 본 시스템은 기존의 독립적인 전자상거래 중개자들과는 달리 IMP에서 요구되는 종속적인 서비스 결합 및 자동 실행을 지원하고 있다. 즉, 종속된 서비스들은 워크플로우 프로세스로 표현되고 이 프로세스들은 워크플로우 엔진에 의해 자동 처리된다. 많고 다양한 이질적 분산 환경에서 개방성과 상호운용성을 보장하기 위해 제안된 시스템은 차세대 웹 표준언어인 XML을 기반으로 설계되고 구현되었다. 구현된 중개자는 웹(Web) 프로그래밍을 위해 ASP+를 사용하였고, 워크플로우 엔진을 위해 COM+를 사용하였다. ASP+와 다양한 컴포넌트들 사이의 자료 교환은 XML을 사용함으로써 시스템 독립성과 시스템 유연성을 향상시켰다.

IMP를 운용하기 위해서는 하나 이상의 특정 응용분야를 선

택하고 관련 서비스들의 메타 데이터베이스를 구축해야 한다. 그렇지만 본 논문에서는 실제 데이터베이스는 구축하지 않고 이러한 마켓플레이스에서 중요한 역할을 수행하는 서비스 중개자에 대한 구현기법만 기술하고 있다. 실제 데이터베이스 구축 및 이에 대한 성능분석 연구가 향후 필요하다.

부 록

1. dsm.dtd

```
<!ELEMENT dsm (general, access, domain)>
<!ATTLIST dsm
          id CDATA " ">
<!ELEMENT name (#PCDATA)>
<!ELEMENT general (person+, creation, version, rights)>
<!ELEMENT person (affiliation?, address?, url?, email)>
<!ATTLIST person
   type (author | coauthor | provider | others) "others"
   lastname CDATA " "
   firstname CDATA " "
   middle CDATA " ">
<!ELEMENT affiliation (#PCDATA)>
<!ELEMENT address (street?, city?, zipcode?, country?, state?,
          phone * )>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT zipcode (#PCDATA)>
<!ELEMENT country (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ATTLIST phone
   type (work | fax | home) "work">
<!ELEMENT url (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT creation ANY>
<!ATTLIST creation
   year CDATA " "
   month CDATA " "
   day CDATA " "
   hour CDATA " "
   minute CDATA " "
   second CDATA " ">
<!ELEMENT rights (owner, group, other) >
<!ELEMENT owner (#PCDATA)>
<!ELEMENT group (#PCDATA)>
<!ELEMENT other (#PCDATA)>
<!ELEMENT access (file | db)>
<!ELEMENT file (addr, type?)
<!ATTLIST file
   decomposition (none | gunzip | decompress) "none"
   format (ascii | binary | others) "others"
   protocol (http | ftp | others) "http">
<!ELEMENT addr (#PCDATA)
<!ELEMENT type (array | list | matrix | file | #PCDATA)>
<!ELEMENT array (type)>
<!ATTLIST array
   rows CDATA " ">
<!ELEMENT list (type) * >
<!ELEMENT matrix (type)>
<!ATTLIST matrix
   rows CDATA " "
   cols CDATA " ">
<!ELEMENT db (addr, user?, password?, query, driver)>
<!ELEMENT user (#PCDATA)>
<!ELEMENT password (#PCDATA)>
<!ELEMENT query (#PCDATA)>
<!ELEMENT driver (#PCDATA)>
<!ELEMENT domain (category, description, keyword*, abstract?,
          paper?)>
<!ELEMENT category (attribute*)>
<!ATTLIST category
   name CDATA" ">
<!ELEMENT attribute (value+, description?)>
<!ATTLIST attribute
   name CDATA " ">
<!ELEMENT value (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT keyword (#PCDATA)>
<!ELEMENT abstract (#PCDATA)>
<!ATTLIST abstract
   href CDATA " ">
<!ELEMENT paper (#PCDATA)>
<!ATTLIST paper
   href CDATA " ">
```

2. msm.dtd

```
<!ELEMENT msm (general, access, interface, domain)>
<!ATTLIST msm
   id CDATA " ">
<!ELEMENT name (#PCDATA)>
<!ELEMENT general (person+, creation, version)>
<!ELEMENT person (affiliation?, address?, url?, email)>
<!ATTLIST person
   type (author | coauthor | provider | others) "others"
   lastname CDATA " "
   firstname CDATA " "
   middle CDATA " ">
<!ELEMENT affiliation (#PCDATA)>
<!ELEMENT address (street?, city?, zipcode?, country?, state?,
          phone * )>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT zipcode (#PCDATA)>
<!ELEMENT country (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ATTLIST phone
   type (work | fax | home) "work">
<!ELEMENT url (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT creation ANY>
<!ATTLIST creation
   year CDATA " "
   month CDATA " "
   day CDATA " "
   hour CDATA " "
   minute CDATA " "
   second CDATA " ">
<!ELEMENT version (#PCDATA)>
<!ELEMENT access ANY>
<!ATTLIST access
   src CDATA " "
   type CDATA " "
   language (VB | C++ | Java | C | C# | VBscript |
          JavaScript|others) "others">
<!ELEMENT interface (call, input, output, precond?, postcond?,
          helproutine?)>
<!ELEMENT call(#PCDATA)>
<!ELEMENT input (parameter) * >
<!ELEMENT output (parameter) * >
<!ELEMENT parameter (type, description?)>
<!ATTLIST parameter
   name CDATA " ">
<!ELEMENT type (array | list | matrix | file | # PCDATA)>
<!ELEMENT array (type)>
<!ATTLIST array
   rows CDATA " ">
<!ELEMENT list (type) * >
```

```
<!ELEMENT    matrix      (type)>
<!ATTLIST    matrix
               rows CDATA " "
               cols CDATA " ">
<!ELEMENT    file(addr, type?)
<!ATTLIST    file
               decomposition (none|gunzip|decompress) "none"
               format (ascii|binary|others) "others"
               protocol (http|ftp|others) "http">
<!ELEMENT    addr        (#PCDATA)
<!ELEMENT    precond     (#PCDATA)>
<!ELEMENT    postcond    (#PCDATA)>
<!ELEMENT    helproutine   (access)>
<!ATTLIST    helproutine
               name    CDATA " "
               id      CDATA " ">
<!ELEMENT    domain      (category, description, keyword *, abstract?,
                         paper?)>
<!ELEMENT    category    (attribute * )>
<!ATTLIST    category
               name CDATA" ">
<!ELEMENT    attribute   (value+, description?)>
<!ATTLIST    attribute
               name CDATA " ">
<!ELEMENT    value       (#PCDATA)>
<!ELEMENT    description (#PCDATA)>
<!ELEMENT    keyword     (#PCDATA)>
<!ELEMENT    abstract    (#PCDATA)>
<!ATTLIST    abstract
               href CDATA " ">
<!ELEMENT    paper       (#PCDATA)>
<!ATTLIST    paper
               href CDATA " ">
```

3. epm.dtd

```
<!ELEMENT    epms        (epm * )>
<!ELEMENT    % activity  'task | serial-block | for-each -block |
                         condition-block'>
<!ELEMENT    epm         (% activity ; * )>
<!ATTLIST    epm
               id CDATA " ">
<!ELEMENT    name        (#PCDATA)>
<!ELEMENT    service     (data | method)>
<!ELEMENT    task        (input, output)>
<!ATTLIST    task
               name CDATA " ">
<!ELEMENT    input       (parameter) * >
<!ELEMENT    output      (parameter) * >
<!ELEMENT    parameter   (type, description?)>
<!ATTLIST    parameter
               name CDATA " ">
<!ELEMENT    type        (array | list | matrix | file | # PCDATA)>
<!ELEMENT    array       (type)>
<!ATTLIST    array
               rows CDATA " ">
<!ELEMENT    list(type) * >
<!ELEMENT    matrix      (type)>
<!ATTLIST    matrix
               rows CDATA " "
               cols CDATA " ">
<!ELEMENT    file(addr, type?)
<!ATTLIST    file
               decomposition (none | gunzip | decompress) "none"
               format (ascii|binary|others) "others"
               protocol (http|ftp|others) "http">
<!ELEMENT    addr        (#PCDATA)
<!ELEMENT    serial-block      (% activity ; * )>
<!ELEMENT    for-each-block    (% activity ; * )>
<!ELEMENT    condition-block   (condition, % activity ; * )>
<!ELEMENT    condition (lhs-expr, comp-opr, rhs-expr)>
<!ELEMENT    lhs-expr (#PCDATA)>
<!ELEMENT    comp-opr EMPTY>
<!ATTLIST    comp-opr
               type (EQUAL | GREATER | LESS | GEQ | LEQ |
                     NOTEQUAL)>
<!ELEMENT    rhs-expr      ((int | string), (operator, (int | string)) * >
<!ELEMENT    operator    ("+" | "-" | "*" | "/" | "%")>
<!ELEMENT    int         (#PCDATA)>
<!ELEMENT    string      (#PCDATA)>
```

참 고 문 헌

[1] Global News & Analysis for Application Service Providers, INT Media Group, Inc., http : //www.aspnews.com and http : //www.internet.com.

[2] G. Tamm and O. Gunther, "Business Models for ASP Marketplaces," Proc. 12th European Conference on Information Systems (ECIS 2000), Vienna, 2000.

[3] D. J. Abel, "Spatial Internet Marketplaces : A Grand Challenge?," Proc. 5th Int. Symposium on Spatial Databases (SSD'97), 1262 LNCS, Springer-Verlag, 1997.

[4] 이용주, "인터넷 마켓플레이스를 위한 XML 기반 계획 에이전트의 설계와 구현", 정보처리학회논문지D, 제8권 제3호, pp. 211-220, 2001.

[5] A. Dogac, I. Durusoy, S. Arpinar, E. Gokkoca, N. Tatbul, and P. Koksal, "An Electronic Marketplace through Agents," Int. Workshop on Component-based Electronic Commerce, July 25, 1998.

[6] PersonaLogic, http : //www.personalogic.com.

[7] Amazon, http : //www.amazon.com.

[8] WfMC, "WfMC(Workflow Management Coalition) Standard Documents," Technical Report, Workflow Management Coalition, November, 1998.

[9] S. Holzner, XML Complete, McGraw-Hill, New York, 1998.

[10] H. A. Jacobsen and O. Guenther, "Middleware for Software Leasing over the Internet," ACM Conf. on Electronic Commerce (EC-99), Denver, Colorado, USA, November, 1999.

[11] D. J. Abel, V. Gaede, K. L. Taylor, and X. Zhou, "SMART : Towards Spatial Internet Marketplaces," GeoInformatica, Vol.3, Issue 2, Kluwer Academic Publishers, pp.141-164, 1999.

[12] H. K. Bhargava, R. Krishnan, and R. Mueller, "Decision Support on Demand : Emerging Electronic Markets for Decision Technologies," Decision Support Systems 19, Elsevier Science, pp.193-214, 1997.

[13] H. Li, "XML and Industrial Standards for Electronic Commerce," Knowledge and Information Systems 2, pp.487-497, 2000.

[14] 손진현, 오석균, 이윤준, 김명호, "고성능 분산 워크플로우를 위한 선형 계획법 기반의 워크플로우 작업 할당 방법", 정보과학회논문지 : 데이타베이스, 제27권 제9호, pp.549-557, 2000.

[15] W3 Consortium, "Document Object Model(DOM)," http :

//www.w3.org/DOM/.

[16] M. Mussa and S. Rosen, "Monopoly and Product Quality," Journal of Economic Theory, 18, 2, pp.301-317, August, 1978.

[17] N. Marchand and H. A. Jacobsen, "An Economic Model to Study Dependencies Between Application Software Venders and Application Service Provides," 3nd Berlin Internet Economic Workshop, Berlin, Germany, May 26-27, 2000.

[18] Microsoft.net Home Page, http : //www.microsoft.com/net/.

[19] J. Rumbaugh, et. al., Object-Oriented Modeling and Design, Prentice-Hall, 1991.

이 용 주

e-mail : yongju@sangju.ac.kr

1983년 울산대학교 산업공학과(공학사)

1985년 한국과학기술원 산업공학과(공학석사)

1997년 한국과학기술원 정보및통신공학과 컴퓨터공학전공(공학박사)

1985년~1989년 시스템공학연구소 연구원

1987년~1988년 일본 IBM TRL 연구소 연구원

1989년~1994년 삼보컴퓨터 근무

1997년~1998년 한국과학기술원 Post Doc.

1998년~현재 상주대학교 컴퓨터공학부 조교수

관심분야 : 웹 데이타베이스, 정보검색, 공간 데이타베이스