시맨틱 웹 기반 분산 아키텍쳐 설계

김정석[a], 이재호[b]

서울시립대학교 전자전기컴퓨터공학부
서울특별시 동대문구 전농동 90번지 서울시립대학교 130-743
[a] *Tel: +82-02-2210-2329, E-mail: jskim@ece.uos.ac.kr*
[b] *Tel: +82-02-2210-2629, E-mail: jaeho@ece.uos.ac.kr*

요약

기존 웹 서비스에서 구현하고 있는 n-Tier 분산 아키텍쳐에 대해 시맨틱 웹 서비스를 제공하기 위해 변화되는 내용을 살펴본다. 시맨틱 웹에서는 사람만이 알아보고 분석할 수 있는 형태의 데이터가 아닌 컴퓨터가 이해할 수 있도록 데이터를 표현하게 된다. 컴퓨터가 이해하기 위한 데이터는 단순히 RDBMS에 대한 질의로 이루어지는 것이 아니라 온톨로지에 의한 표현이 필요하게 된다. 기존의 웹은 대부분 데이터를 저장하기 위해 RDBMS를 이용하고 있으며 온톨로지 표현은 이러한 데이터를 기반으로 작성된다면 시맨틱 웹 구성에 많은 도움이 된다.

또 다른 기존의 웹과 시맨틱 웹의 차이점으로 들 수 있는 것은 에이전트가 활용된다는 점이며 이러한 에이전트의 등장으로 인해 각 에이전트의 통신, 지식의 공유와 같은 여러가지 다른 요소가 고려되어야 한다. 이 논문에서는 시맨틱 웹을 구현하기 위해 에이전트 간의 통신 방법과 지식 표현을 위한 방법에 중점을 두어 시맨틱 웹 분산 아키텍쳐를 제안한다.

주제어:

[†]시맨틱 웹, 분산 아키텍쳐, 에이전트

서론

월드 와이드 웹에는 방대한 양의 데이터가 산재해 있다. 간단한 데이터만을 가공하고 추출하려 한다면 클라이언트에서 모든 데이터를 받아 프리젠테이션 로직(GUI), 네비게이션(Navigation), 비즈니스 로직, 데이터 조작 등의 일련의 작업들을 클라이언트에서 처리하게되는 비대(Fat) 클라이언트와 데이터 저장, 트랜잭션 모니터 등을 수행하는 서버로 구성된 2-Tier 아키텍쳐로도 처리가 가능하였다[1]. 그러나 처리할 데이터가 방대해지고 복잡해짐에 따라서 모든 데이터 처리 작업을 클라이언트에서 수행하기에는 많은 어려움이 생기게 되어 n-Tier의 아키텍쳐를 사용하게 되었다.

[†] 본 연구는 첨단정보기술 연구센터를 통하여 과학재단의 지원을 받았음.

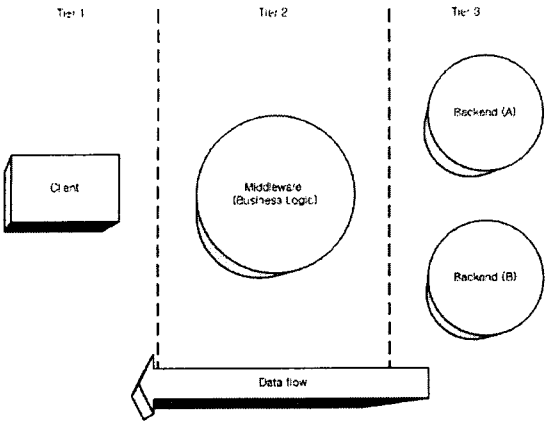

그림 1 3-Tier 클라이언트/서버 아키텍처

n-Tier 아키텍처에서는 프리젠테이션 로직만을 갖는 가벼운(thin) 클라이언트와 비즈니스 로직과 데이터 접근 로직으로 구성된 컴포넌트를 작성하여 여러 응용 시스템에 분산하고 후단에 흔히 데이터 서버라고 불리우는 데이터 저장용 서버를 두게 되어 보통 3 Tier 이상의 응용 시스템을 구성하여 사용하고 있다. 각 로직별로 컴포넌트를 구성하여 별도의 시스템에서 처리하도록 설계한 분산 아키텍처는 사용자에 대한 빠른 반응성과 데이터 처리의 효율성 증대뿐만 아니라 많은 확장 가능성을 열어주었다.

기존의 분산 아키텍처에서 시맨틱 웹으로 확장하려 할 때 추가되는 내용을 소개하고자 한다.

시맨틱 웹 분산 아키텍처

시맨틱 웹의 정의

시맨틱 웹의 의미를 정확하게 단정짓기는 어렵지만 큰 맥락에서 살펴본다면 웹에 산재되고 중복되는 방대한 양의 데이터를 메타 데이터를 이용하여 표현하고 이를 컴퓨터가 알아볼 수 있도록 하여 사람에게 편의성을 제공하고자하는 것이다[2][9].

컴퓨터가 알아보기 위한 데이터를 위해서는 메타 데이터를 이용한 온톨로지 표현이 주가 되고 있으며 이를 이해하는 에이전트는 시맨틱 웹 분산 아키텍처의 필수 요소가 된다.

시맨틱 웹에서의 데이터 표현

기존 웹에서 시맨틱 웹으로 전환한다고 할 때 가장 많이 차이가 나는 부분이 데이터의 표현 부분이다. 기존의 웹에서 데이터는 단순히 RDBMS와 같은 영구 저장 장치에 보존하고 그 데이터를 읽어서 사람이 알아볼 수 있도록 화면에 배치하는 정도의 가공만이 필요했다. 그러나 시맨틱 웹에서는 데이터에서 에이전트에 의한 추론이 가능하도록 온톨로지적인 표현을 지원해야한다. 데이터를 온톨로지로 표현하기 위해서는 DAML+OIL 혹은 OWL을 이용할 수 있다. 기존 RDBMS에 저장된 내용에서 다시 온톨로지로 표현한다는 것은 쉬운 일은 아니지만 충분한 가능성이 있다.

표 1 RDBMS에 존재하는 데이터

상품명	가격	제조회사
레드와인	12500	와인회사
화이트와인	13000	와인회사

실제 데이터베이스에 저장된 정보가 표 1과 같다면 이 내용을 완전하지는 않지만 웹 온톨로지 언어인 OWL의 내용으로 표현한다면 다음 표 2와 같이 된다.

표 2 OWL을 이용한 데이터 표현

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf>
    <owl:Restriction>
     <owl:onProperty
rdf:resource="#hasMaker"/>
      <owl:onProperty
rdf:resource="#hasPrice"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasMaker">
  <rdf:type rdf:resource="http://www.w3.org/20
02/07/owl#FunctionalProperty"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="price"/>
<owl:DatatypeProperty rdf:ID="priceValue">
  <rdfs:domain rdf:resource="price"/>
  <rdfs:range
   rdfs:resource="http://www.w3c.org/2000/10
   /XMLSchema#postiveInteger"/>
</owl:DatatypeProperty>
<price rdf:ID="레드와인가격">
  <priceValue>12500</priceValue>
</price>
<price rdf:ID="화이트와인가격">
  <priceValue>13000</priceValue>
</price>
<Maker rdf:ID="와인회사"/>
<Wine rdf:ID="레드와인">
  <hasMaker rdf:resource="#와인회사"/>
  <hasPrice rdf:resource="#레드와인가격"/>
</Wine>
<Wine rdf:ID="화이트와인">
  <hasMaker rdf:resource="#와인회사"/>
  <hasPrice rdf:resource="#화이트와인가격"/>
</Wine>
```

표 2의 OWL의 표현에서 보면 각 데이터에 대한 정의가 있고 그 뒤에 정의를 이용하여 실제 데이터를 생성하여 사용하는 것을 알수 있다. OWL은 객체, 속성과 그들간의 관계를 정의한 부분과 실제 인스턴스를 생성한 두 부분으로 나뉘어짐을 알 수 있으며 RDBMS의 저장된 내용은 OWL의 인스턴스 표현 부분임을 알 수 있다. 이러한 점을 이용하여 인스턴스는 정의하지 않은 상태의 OWL을 별도로 저장하고 이를 토대로 RDBMS의 내용을 OWL의 문법에 맞는 형태로 추출해주는 OWL 변환기를 충분히 제작할 수 있게 된다. OWL 또한 XML 데이터이므로 현존하는 XML과 DB간의 매핑 기술을 이용할 수 있다[3].

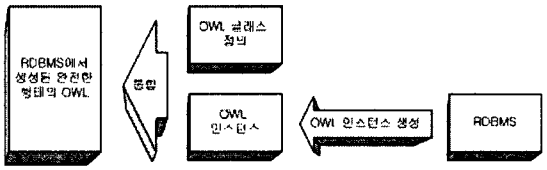

그림 2 RDBMS to OWL 변환기 모식도

에이전트의 추론

온톨로지로 표현된 데이터들을 사용하기 위해서는 에이전트의 온톨로지 추론 기능이 필수적이다. OWL과 같은 언어를 이용하여 표현된 언어는 FOL로 변환이 쉬우므로 Prolog와 같은 로직 언어를 이용하여 추론할

수 있도록 에이전트에 이러한 기능을 추가하여야 한다. 그러나 분산 아키텍쳐에서 각 에이전트마다 온톨로지 추론 기능을 가지고 있다는 것은 빠른 응답과 데이터 처리의 효율성 면에서 그리 바람직한 것은 아니다. 때문에 온톨로지 추론 엔진은 에이전트 내부가 아닌 또 다른 하나의 서비스를 구성해주는 것이 바람직하다. 이때 추론 서비스를 제공하는 서버는 에이전트의 요구에 따라 민첩하게 반응하여야하므로 RPC(Remote Call Procedure)를 이용하는 것이 바람직하며 온톨로지 데이터베이스는 앞서 제시한 RDBMS로부터 OWL로의 변환기의 출력 자료가 될 수 있다.

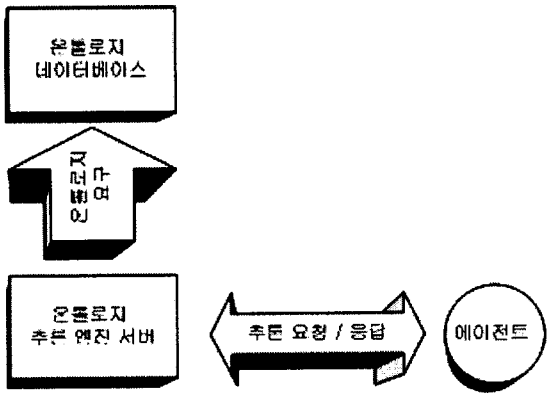

그림 3 에이전트를 위한 추론 서비스

추론 엔진을 에이전트와 분리하게 되면 처리 성능면에서 많은 이점을 얻을 수 있다. 추론 대상이 되는 데이터가 많아진다면 추론 엔진 자체를 병렬 처리 시스템으로 구성하여 추론 엔진의 성능을 높힐 수 있다[4].

에이전트 간의 통신

시맨틱 웹을 구성하기 위한 에이전트는 단일 에이전트로는 의미가 없다. 사용자에 해당하는 에이전트, 서비스 제공자의 에이전트들이 서로 통신을 하며 복잡하고 다양한 사용자의 작업을 처리하기 위해 멀티 에이전트 프레임워크가 필요하게 된다[5].

와인을 판매하는 상점과 와인 잔을 판매하는 상점의 에이전트가 있고 사용자는 레드 와인을 마실 때 사용하는 와인 잔을 구매한다는 시나리오를 가정하여 본다면, 와인 잔을 판매하는 상점의 에이전트의 온톨로지에는 와인에 대한 정보가 없으므로 와인을 판매하는 상점의 에이전트에게 데이터를 요구하는 상황이 발생하게 되는 것이다.

이러한 에이전트 간의 통신을 위해서는 ACL (Agent Common Language)를 이용하여 자바로 구현된 JADE(Java Agent Development Framework)가 있다[6].

JADE는 자바로 구현된 소프트웨어 프레임워크로 FIPA(Foundation for Intelligent Physical Agent)의 명세를 따르며 디버깅과 배포에 관한 도구를 제공한다. 각 에이전트마다 ACL 메시지 큐를 생성, 관리하며 블로킹, 폴링, 타임 아웃, 패턴 매칭과 같은 방법들을 조합하여 큐의 메시지들에 접근한다. JADE를 이용한 에이전트 통신 방법은 뒤에 설명할 TAGA에서 구현하고 있으며 이를 확장 구현하고자 한다.

시맨틱 웹 분산 아키텍쳐 모델

앞서 제시한 내용을 토대로 시맨틱 웹을 위한 아키텍쳐를 구성하게 되면 기존 웹의 분산 아키텍쳐에 에이전트를 위한 구조를 추가하는 것으로 시맨틱 웹 분산 아키텍쳐 모델을 구성할 수 있게 된다.

시맨틱 웹 분산 아키텍쳐를 서비스

제공자와 사용자의 관점에서 모식도를 표현하면 그림 4와 같은 표현이 된다.

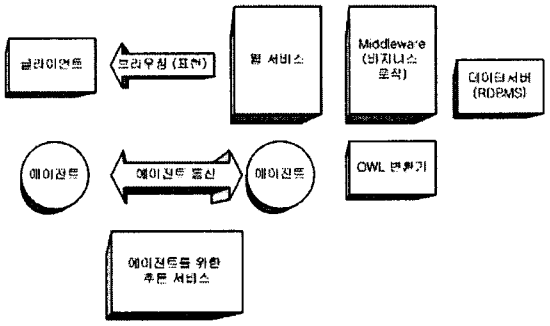

그림 4 시맨틱 웹 분산 아키텍처

그림 4의 그림에서 살펴보면 각 에이전트의 통신 부분이 매우 간결하게 표시되어있다. 각 에이전트간의 통신에서는 서비스를 제공하는 에이전트와 서비스를 요구하는 사용자 에이전트 간에는 명확히 다른 방법으로 구현이 되어야 하며 이는 TAGA(Tavel Agent Game in Agentcities)에서 사용하는 방법과 비슷한 형태로 통신을 하도록 구성할 수 있다. TAGA의 통신 방법은 그림 5와 같다[7].

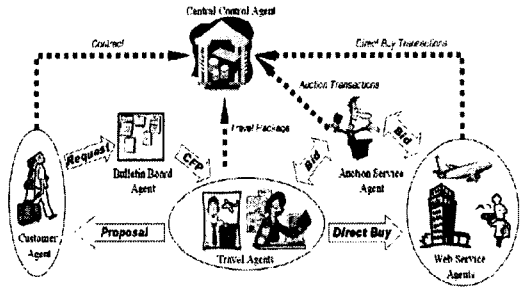

그림 5 TAGA의 프레임워크 모식도[8]

TAGA에 따르면 각 에이전트 간의 통신을 위해 앞서 제시한 JADE를 이용하고 있다.

통신 방법은 고객의 에이전트가 필요한 내용을 블랙 보드 에이전트에 전달하면 블랙 보드 에이전트는 상점 에이전트에 이 사항을 브로드캐스팅 해주고 이를 받은 상점 에이전트는 고객 에이전트의 요구에 맞는 제안서를 직접 고객의 에이전트에 제출하게 된다. 이를 고객 에이전트가 수락하게 되면 에이전트끼리 협상을 하게 되는 것이다.

이러한 에이전트 통신 기법을 변형하여 앞서 제시한 추론 서비스를 이용하는 에이전트를 기준으로 다시 표현한다면 그림 6과 같은 형태가 된다.

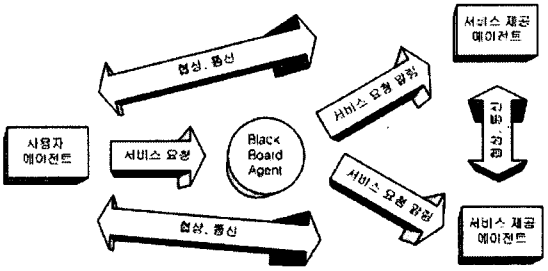

그림 6 에이전트 간의 통신 방법 모식도

결론

본 논문에서는 현재 많은 관심과 논의가 진행되고 있는 시맨틱 웹을 멀티 에이전트 응용 프레임웍에 도입하기 위한 방안을 기존 웹의 분산 시스템과 비교하여 제시하였다. 시맨틱 웹의 도입은 멀티 에이전트 아키텍처의 개방성 및 확장성, 기존 시스템과의 통합성, 시스템 지능화에 크게 기여할 것으로 기대된다.

참고 문헌

[1] Henri Jubin, Jurgen Friedrichs. (1999). Enterprise JavaBeans by Example: Prentice-Hall.

[2] CNET Korea (2002). "WWW에서 시맨틱 웹으로" 마이크로소프트웨어, 2002년 4월호, pp. 244-251.

[3] 이주한. (2002). "관계형 데이터베이스 스키마를 위한 XML 인터페이스 설계 및 구현", 석사학위논문, 서울시립대학교.

[4] Jose Cunha, Rui Marques.(1996). "PVM-Prolog: A Prolog Interface to PVM", Universidae Nova de Lisboa.

[5] 강기영, 장지훈, 최중민. (1998). "Java를 이용한 멀티 에이전트 기반 구조", 인지과학회 논문지, 9권 2호, pp 25-36.

[6] "Java Agent Development Framework", http://jade.cseit.it

[7] "Travel Agent Game in Agentcities", http://taga.umbc.edu/taga2

[8] Youyong Zou, Tim Finin, Li Ding, Harry Chen, and Rong Pan.(2002). "TAGA: Trading Agent Competition in Agentcities", University of Maryland.

[9] 이재호 (2003), "시맨틱 웹의 온톨로지 언어", 정보과학회지, 21권3호, pp. 18-27.