유비쿼터스 DSS 포털을 위한 지능형 온톨로지 관리 시스템: - u-Fulfillment 도메인 중심 -

이현정[a], 이건창[b], 손미애[c]

[a] 한국과학기술원 테크노경영연구소
130-722, 서울시 동대문구 청량리동 207-43
Tel: +82- 2-958-3916, E-mail:hjlee5249@yahoo.com

[b] 성균관대학교 경영학부 교수
110-745, 서울시 종로구 명륜동 3가 53번지
Tel: +82-2-760-0505, E-mail:Kunchanglee@gmail.com

[c] 성균관대학교 시스템경영공학과 조교수
440-746, 경기도 수원시 장안구 천천동 300
Tel: +82-31-290-7605, E-mail: myesohn@msn.com

Abstract

본 연구에서는 유비쿼터스 환경에서 시시각각으로 변하는 고객의 요구사항을 만족시키기 위해 상호 협력을 시도하는 다중 에이전트들이 의사결정을 수행할 경우 발생할 수 있는 충돌의 해결을 지원하는 지능적인 온톨로지 관리 모듈 (intelligent-Ontology Management Module, i-OMM)과 다중에이전트 관리 모듈(Multi-agent Coordination Module, MACM)을 포함하는 u-DSS 포털을 제안한다. 개별 에이전트들은 온톨로지와 문제해결을 위한 프로시저 등을 이용해 자신의 문제를 해결하는 것을 기본으로 한다. 그러나 다른 에이전트들과 협력을 통해 문제를 해결해야 할 경우에는 먼저 개별 에이전트들이 보유한 각 온톨로지의 통합 및 데이터간의 충돌 해결이 요구된다.

i-OMM은 개별 에이전트들이 보유하고 있는 이질적인 온톨로지를 통합하여, 문제해결을 위한 하나의 통합된 새로운 동적 온톨로지 뷰 (integrated ontology view, IOV)의 생성을 지원한다. 온톨로지 통합과제에서 생성된 IOV는 사례로 저장되고 유사한 문제 해결에 재사용된다. MACM은 고객 에이전트들의 요구사항 변화에 따라 관련 개별 에이전트들 간의 데이터 충돌을 해결하여 에이전트들의 의사결정과정을 지원한다. 따라서 i-OMM과 MACM을 이용한 유비쿼터스 컴퓨팅 환경에서 분산된 에이전트들의 협력적 문제 해결을 지원하는 시스템을 유비쿼터스 의사결정지원 시스템 포탈 (ubiquitous decision support system, u-DSS Portal)이라고 지칭한다. 본 연구에서 제안된 알고리즘의 활용 대상은 고객, 판매자, 제조업체, 및 배송업체의 배송차량들의 에이전트들로 구성된 u-fulfillment 시스템으로 한다.

Keywords: *유바쿼터스 의사결정지원시스템, 지능적인 온톨로지, 다중에이전트시스템, 동적 통합 온톨로지*

1. 서론

유비쿼터스 컴퓨팅 환경의 장점은 시시각각으로 변하는 정보를 언제 어디서나 수집하는 것이 가능하다는 것이다. 그러나 정보의 수집이상으로 중요한 것이 정보의 활용이다. 즉 다량의 양질의 정보가 의사결정자들에게 유효한 영향을 미칠 때 정보로서의 가치를 인정받는다. 유비쿼터스 컴퓨팅 환경에서는 의사결정자들이 장소나 시간에 구애받지 않고 의사결정을 수행하도록 지원할 수 있어야 하고 이에 대한 다양한 연구가 이루어져 왔다.

유비쿼터스 환경하에서의 효과적인 의사결정 지원을 위해서는 소프트웨어 에이전트들이 협력적으로 문제를 해결하기 위해 필요한 이질적인 데이터나 정보를 막힘 없이 활용할 수 있어야 한다. 본 연구에서는 유비쿼터스 컴퓨팅 환경에서 분산된 에이전트들의 협력적 문제 해결을 위한 시스템인 유비쿼터스 의사결정지원 시스템 포탈 (ubiquitous decision support system portal, u-DSS portal) 을 제안한다.

u-DSS 포탈 시스템은 다음과 같은 주요 기능들로 구성된다.

첫째, u-DSS 에이전트는 스마트 기기들로부터 전달받은 정보를 해석하여 문제를 인식한 후, 필요하다면 u-DSS 엔진의 다중에이전트들간의 이질적인 온톨로지 통합을 위한 조정을 요청하는 기능을 수행한다.

둘째, 이질적인 온톨로지를 통합하여 의사결정을 지원하기 위해 지능적인 온톨로지 관리 모듈 (***i***ntelligent-***O***ntology ***M***anagement ***M***odule, i-OMM)을 제안한다. i-OMM은 이질적인 온톨로지를 통합하기

위해 온톨로지 구조 비교, 어휘비교 및 의미비교 등을 통해 새로운 온톨로지를 생성하는 방법으로, 이질적인 온톨로지를 통합하여 문제해결을 위한 하나의 통합된 가상 온톨로지 뷰 (integrated ontology view, IOV)를 생성한다. i-OMM에서 IOV의 활용은 먼저 과거에 유사한 문제를 협력적으로 해결한 사례가 있으면, 사례베이스에 저장된 IOV를 탐색하여 문제를 해결한다. 그러나 사례의 활용은 추후 연구에서 다루기로 한다.

셋째, 다중에이전트들이 관여되어 있는 문제를 협력적으로 해결하는 다중에이전트 협력 모듈(multi-agent coordination mechanism, MACM)을 제안한다. MACM은 i-OMM에서 생성된 통합 온톨로지 뷰를 기반으로 다중 에이전트들간의 데이터 충돌 문제를 해결한다.

본 연구에서 가정하는 유비쿼터스 컴퓨팅 환경에서의 의사결정 상황은 다음과 같다. 예를 들어, 에이전트 A가 수행하는 의사결정이 에이전트 B나 C의 상황 변화에 영향을 주고, 에이전트 B나 C의 상황 정보가 에이전트 A에게 실시간으로 전달될 수 있음을 기본으로 한다. 이러한 다중에이전트들간의 문제 해결을 위한 방법 중의 하나는 연관된 에이전트들이 하나의 온톨로지를 공유하는 것이다 [8]. 그러나 공통의 온톨로지 공유 방식은 다음과 같은 문제점들이 내포되어 있다. 첫째, 방대한 양의 공통 온톨로지를 모든 에이전트가 보유해야 하기 때문에 처리 속도에 영향을 줄 수 있다. 둘째, 개별 에이전트가 처한 상황 변화로 인해 공통 온톨로지를 변경해야 할 경우 일관성 있는 변경이 어렵다. 그리고 마지막으로 공유 온톨로지의 완벽성이 만족되지 않아 문제 해결에 어려움이 발생할 수 있다. 따라서 본 연구에서는 기존의 공통 온톨로지의 공유를 통한 협력적 문제해결과 달리 개별 에이전트들이 가진 온톨로지의 동적인 통합을 통한 협력적 문제해결을 지원하는 지능적 온톨로지 관리 모듈인 i-OMM을 제안한다.

i-OMM에 의해 동적인 통합 온톨로지가 생성된 이후 고객의 요구사항 변화에 따른 다중 에이전트들간의 데이터 값 변화에 따른 충돌 해결을 위한 지원이 필요하다. 이를 지원하는 MACM은 에이전트 A, B 와 C간의 온톨로지 통합 이후 요구사항 변화에 따른 데이터 충돌을 해결한다.

본 논문의 구성은 다음과 같다. 2장에서 관련 문헌들을 살펴보고, 3장에서는 u-DSS 포탈의 기본 구조를 간략히 설명하고, 4장에서는 온톨로지 동적 통합 및 데이터 충돌 해결 알고리즘을 제시한다. 5장에서는 u-DSS 포털에서의 지능형 온톨로지 관리를 위한 정보처리의 절차를 살펴본다. 6장에서 u-fulfillment 예제를 통한 u-DSS 포탈의 활용 예를 살펴보고 7장에서 결론 및 향후과제에 대해 살펴보기로 한다.

2. 문헌조사

2.1 유비쿼터스 의사결정지원 시스템

유비쿼터스 의사결정지원시스템은 기본적으로 유동 사용자들의 의사결정을 지원하도록 설계된다 [14]. 유동 사용자들은 기본적으로 정적 사용자와 많은 부분에서 구분된다 [10, 2]. 따라서 유비쿼터스 환경에서의 의사결정지원시스템은 기존의 시스템들과 많은 부분에서 차이가 있다. 예를 들어 의사결정시스템을 사용하고자 하는 유동 사용자들은 관련 데이터를 출력하거나, 그랙픽 화면을 캡쳐하는 등의 작업을 움직이면서 하고자 하나 기존 시스템들은 이를 지원하지 않는다 [1]. 따라서 의사결정시스템에 유동성을 지원하는 시스템에 대한 요구가 증대되고 있다 [7]. 예를 들어 에이전트들이 중앙의 의사결정지원시스템과 협업을 통해 유동 사용자의 동적 의사결정을 지원하기도 한다 [23]. 따라서 유동 사용자의 의사결정지원을 위한 다중 에이전트들의 협업 지원하기 위한 연구가 요구된다 [7, 14]. 본 연구에서는 이를 위한 연구로 동적 통합 온톨로지 및 다중 협력 에이전트에 관해 연구하기로 한다.

2.2 온톨로지

유비쿼터스 환경에서 자율적으로 활동하는 에이전트들이 다른 에이전트들과 협력적으로 문제를 해결하기 위해서는 "공유된 개념"에 근거한 에이전트들간의 커뮤니케이션이 필수적이다. 에이전트가 다른 에이전트들과 통신을 할 수 있어야 한다는 것은 에이전트의 기본 기능중의 하나로서 [3, 9, 22], 본 논문의 연구주제가 아니다.

다음으로 중요한 것이 개념의 공유인데, 이를 위해서는 협력적으로 문제를 해결하는데 관련되어 있는 모든 에이전트들이 하나의 공통 개념을 사용해야 한다는 것이다. 공통된 개념을 사용하지 않을 경우, 에이전트들이 사용하는 개념들간의 의미론적 모순이나 문법적 모순 등이 발생할 수 있기 때문이다. 이를 해결하기 위해 연구가 온톨로지 통합의 분야에서 수행되고 있으며[4, 6, 11, 19, 21], PROMPT, ONION, Chimaera, 및 GLUE등과 같은 다양한 도구 역시 개발되었다 [5, 15, 17, 18]. 또한 최근에는 Pazienza [20]가 온라인 소매에서의 다중언어 처리를 위한 방법으로 CROSSMARC을 개발한 바 있다.

본 연구 역시 온톨로지의 통합을 통해 협력적인 문제 해결을 시도하고 있다는 면에서는 기존 연구의 맥락에서 벗어나지 않으나, 다음과 같은 측면에서 기존 연구와 차별화 된다. 첫째, 서비스로서의 온톨로지 통합이 이루어진다는 것이다. 특정 에이전트로부터의 협력적 문제 해결 수요가 발생하면 그 요구에 맞는 온톨로지가 생성된다는 것이다. 기존의 연구들이 모든 개념을 통합한 하나의 온톨로지를 생성하고자 했던 것과 비교되는

점이다. 둘째, CBR 개념을 채택함으로써, 온톨로지 생성의 효율성을 높인 것 또한 기존의 온톨로지 통합 연구와 차별화되는 점이다. 과거 문제 해결과정에서 생성된 가상 온톨로지 뷰를 하나의 사례로 취급해 사례베이스를 구축함으로써, 유사 가상 온톨로지 뷰를 활용할 수 있도록 하였다.

2.3 다중에이전트

동적 사용자의 의사결정을 지원하기 위해 다중 에이전트의 협력 시스템이 요구된다. 즉 분산 에이전트 환경에서 의사결정에 참여하는 참여자들간의 협업에 의한 문제 해결을 위한 노력이 중요해지고 있다 [12, 13, 16, 24, 25]. 본 연구에서는 이러한 다중 에이전트의 협업을 지원하기 위해 통합 온톨로지 구성과 함께 데이터 변화에 따른 다중 에이전트간의 데이터 충돌을 해결하는 연구를 제안하고자 한다.

3. u-DSS 포탈의 기본 구조

u-DSS 포털의 기본구조는 다음 그림2와 같이 u-DSS 에이전트 모듈, u-DSS의 지능형 온톨로지 관리 모듈 i-OMM, 다중에이전트 협력 모듈 MACM 및 지식정제모듈 KRM으로 이루어 진다. 즉 u-DSS 포탈의 기능은 사용자들이 보유하고 있는 PDA, 무선 인터넷 PC, 스마트 터미널과 같은 스마트 기기가 사용자의 변화된 상황을 인식함과 동시에 활성화 된다. 인식된 상황정보는 u-DSS 에이전트를 통해 u-DSS 엔진에 전달되고 인식된 문제를 각 구성요소들을 통해 해결하는 것으로 한다. 즉 본 연구에서 u-DSS 포탈은 정보의 공유 및 정보 재구성을 위한 구성요소들간의 온톨로지 및 데이터 충돌해결에 초점을 두고 있다. 다음에서 각각의 모듈에 대해 자세히 살펴보기로 한다.

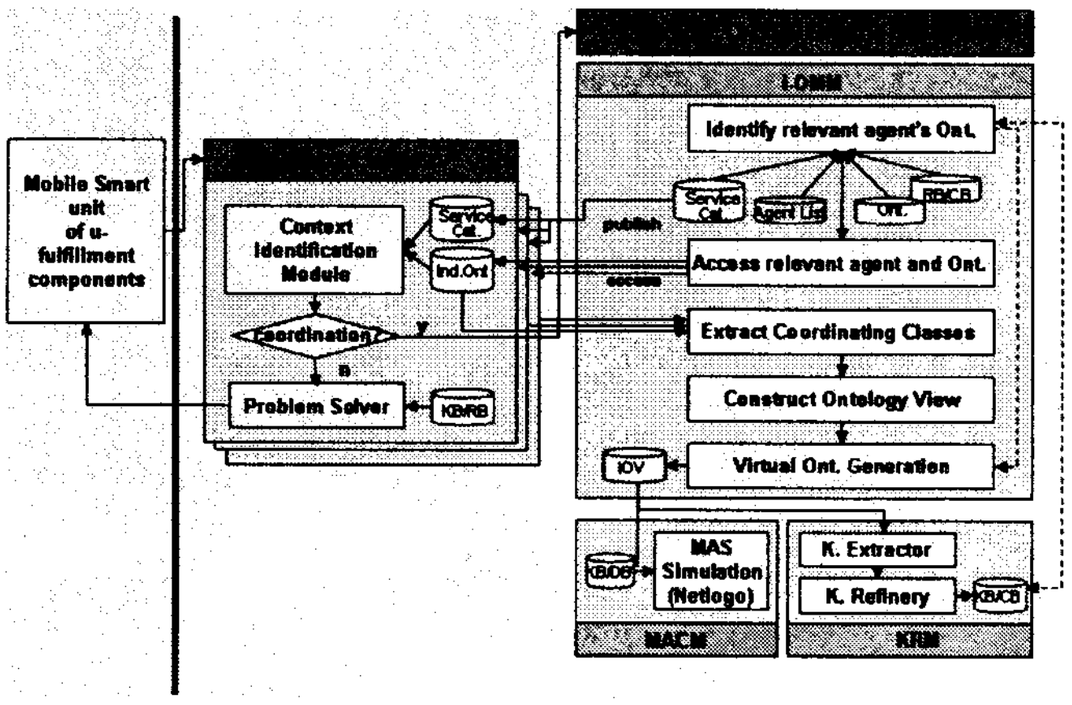

그림 2 – Overall Architecture of the u-DSS portal

3.1 u-DSS 에이전트

u-DSS 에이전트는 스마트 기기들로부터 전달받은 정보를 해석하여 문제를 인식한 후, 필요하다면 u-DSS 엔진에 조정을 요청하는 기능을 수행한다. 이를 위해 구성요소 중 상황인지모듈(Context Identification Module)은 의사결정에 관여된 에이전트들로부터 수신되는 변경된 상황정보를 해석해 개별에이전트 독자적으로 해결할 것인지 아니면 다른 에이전트와의 협력적으로 문제를 해결할지를 판단한다. 이를 위해 CIM은 u-DSS 포탈이 배포한 서비스 카탈로그를 사용한다. 서비스 카탈로그에는 u-DSS 포탈에서 해결할 수 있는 문제와 문제 해결을 위해 입력해야 할 데이터 목록이 명시되어 있다.

3.2 u-DSS 엔진

u-DSS 엔진은 협력이 필요한 개별 에이전트들이 가진 온톨로지로부터 문제 해결 또는 협력에 필요한 온톨로지 구성요소를 추출해 가상의 동적 통합 온톨로지를 생성하는 과정을 지원하며, 다음과 같은 하위 요소를 가진다.

3.2.1 지능형 온톨로지 관리 모듈 (*(intelligent-Ontology Management Module,* i-OMM)

i-OMM은 u-DSS 엔진의 지능형 온톨로지 관리 모듈로서, 유비쿼터스 컴퓨팅 환경에서 작동중인 개별 에이전트들이 협력적으로 문제를 해결해야 할 경우 필요한 가상 동적 통합 온톨로지를 생성하고 관리한다. u-DSS 에이전트 모듈의 요청에 의해 실행되며, 문제 해결을 위해 협력이 필요한 에이전트들이 보유하고 있는 이질적인 온톨로지를 통합하여 현재 문제에 적합한 새로운 가상의 온톨로지 뷰인 IOV를 생성하는 것을 목적으로 한다.

3.2.2 다중 에이전트 협력 모듈 (*Multi-agent Coordination Module, MACM*)

MACM은 i-OMM의 작동 결과 생성된 가상 동적 온톨로지 뷰 IOV를 기반으로 협력한 개별 에이전트들의 데이터 충돌 해결을 지원하는 모듈이다. 즉 i-OMM에서 생성된 동적 통합 온톨로지 뷰인 IOV상에서 고객 요구에 따른 데이터 변화에 의해 발생되는 Instance level간의 데이터 충돌과 이를 해결하기 위한 의사결정을 지원한다. 즉 다중 에이전트 협력 모델에서 개별 에이전트의 이익최대화와 다중 에이전트의 총 이익의 최대화를 고려하여 데이터 값의 충돌을 해결하는 의사결정을 지원한다

3.2.3 지식정제모듈 *(Knowledge Refinery Module, KRM)*

KRM은 문제 해결과정에서 생성된 가상의 온톨로지 뷰를 재사용할 수 있도록 정제하는 역할을 수행한다. 즉, 생성된 IOV를 사례베이스에 저장한 후, i-OMM이 새로운 IOV를 생성해야 할 필요성이 발생한 경우, 4.2절에서 언급한 과정을 거쳐 IOV를 생성하기 전에 사례베이스를 검색해 유사 사례가 있는지를 찾아보게 된다. 사례베이스에 유사 사례가 존재하면 저장된 IOV를 활용해 문제를 해결하고, 유사사례가 존재하지 않으면 새로운 IOV를 생성한다. 이와 동시에 사례베이스에 저장하게 될 IOV의 생성 과정 중에 지식공학자가 클래스들간의

관계를 새롭게 식별한 것이 있다면, 새로운 식별된 관계를 서비스카탈로그에 저장함으로써, 서비스 카탈로그를 발전시키게 된다.

4. 온톨로지 동적 통합 및 데이터 충돌 해결 알고리즘

4.1 온톨로지 동적 통합 알고리즘

온톨로지 동적 통합 알고리즘은 다중에이전트간의 문제해결 과정에서 발생할 수 있는 이질적 온톨로지의 동적인 통합을 매개하는 알고리즘으로 다음과 같은 과정을 통해 이루어진다.

1) 협력적 문제 해결에 관련된 에이전트와 온톨로지 구성요소 식별

가상의 온톨로지 뷰를 생성하는 첫 번째 단계는 해당 문제의 해결을 위해 협력해야 할 에이전트에는 어떤 것이 있는지를 식별하는 것이다. 다음 단계로는 식별된 에이전트가 가진 온톨로지 중 관련된 개념(클래스)을 추출하는 것이다. 이 단계를 자동화하기 위해서는 에이전트에 의한 자연어 처리가 필수적이지만, 현재의 자연어 처리 수준을 고려하여 본 연구에서는 자동화가 아닌 '진화(evolution)'의 방식을 채택하기로 한다.

협력적 문제 해결을 위해 관련된 에이전트와 온톨로지를 추출하기 위해 본 연구에서는 표1과 같은 서비스 카탈로그 (service catalogue)를 기본 가정으로 한다. 기본적으로 서비스 카탈로그는 지식공학자에 의해 에이전트들의 온톨로지 구성요소가 정의된다.

서비스 카탈로그는 서비스 계층과 참조계층으로 이루어져 있다.

- ***서비스 계층 (Service Layer):*** 서비스 계층은 서비스를 요청한 에이전트가 선택한 서비스의 명칭과 서비스를 받기 위해 전송한 입력자료들로 구성된다.
- ***참조계층 (Reference layer):*** 개별 에이전트가 전송한 입력자료를 근거로, 상호 연관되어 있는 클래스들을 클래스베이스로부터 탐색하여 참조계층을 구성한다. 즉 협력적으로 문제해결을 해야 하는 에이전트들의 목록과 클래스들간의 관계를 표현한다. 클래스들간의 관계에는 둘 또는 그 이상의 클래스가 동일한 클래스임을 나타내는 '***equivalentClass***'로 둘 또는 그 이상의 클래스가 배타적인 관계가 있음을 표현하는 '***disjointWith***'로 표현된다. '***equivalentClass***'는 같은 의미임에도 불구하고 개별 에이전트들이 사용하는 용어가 다를 경우 발생하는 의미적 불일치를 해결한다.

서비스 카탈로그는 표1과 같은 계층구조를 가지며, 서비스 카탈로그의 내용은 u-DSS의 문제 해결 결과를 반영하여 지속적으로 개선될 수 있다. 본 연구에서 카탈로그의 진화과정은 제외하기로 한다.

표 1 – 서비스 카탈로그의 구조 및 예

```
<serviceCatalogue>
  <serviceName>getChangedDeliveryCondition
    <serviceLayer>                                                    ┐
      <serviceRequestAgent>userName</serviceRequestAgent>             │
      <inputData>                                                     │
        <changedCondition>deliveryAddress</changedCondition>          │ Service
        <changedCondition>deliveryDate</changedCondition>             │ Layer
      </inputData>                                                    │
    </serviceLayer>                                                   ┘
    <referenceLayer>                                                  ┐
      <className>DeliveryAddress                                      │
        <equivalentClass>deliveryLoc                                  │
          <classOwner>                                                │
            <agentName>deliveryVanAgent</agentName>                   │
            <ontURL>http://www.cjlogis.com/deliveryVan</ontURL>       │
          </classOwner>                                               │
        </equivalentClass>                                            │
        <equivalentClass> deliveryDestination                         │
          <classOwner>                                                │
            <agentName> sellerAgent </agentName>                      │
            <ontURL> http://www.cjmall.com/customer/orderInfo</ontURL>│
          </classOwner>                                               │
        </equivalentClass>                                            │ Reference
      </className>                                                    │ Layer
      <className>DeliveryDate                                         │
        <equivalentClass>deliverySschdeule                            │
          <classOwner>                                                │
            <agentName>deliveryVanAgent</agentName>                   │
            <ontURL>http://www.cjlogis.com/deliveryVan</ontURL>       │
          </classOwner>                                               │
        </equivalentClass>                                            │
        <equivalentClass> deliveryTime                                │
          <classOwner>                                                │
            <agentName> sellerAgent </agentName>                      │
            <ontURL> http://www.cjmall.com/customer/orderInfo</ontURL>│
          </classOwner>                                               │
        </equivalentClass>                                            │
      </className>                                                    │
    </referenceLayer>                                                 ┘
  </serviceName>
  <serviceName/>
</serviceCatalogue>
```

2) 온톨로지 구성요소의 추출 및 IOV 생성

서비스 카탈로그를 이용하여 해결해야 할 문제를 파악하고, 관련된 에이전트들과 클래스들을 식별하였다. 개별 에이전트가 가진 온톨로지 중 해결하여야 할 문제와 관련된 온톨로지만을 식별한 것을 다음 그림 3에 도시하였다.

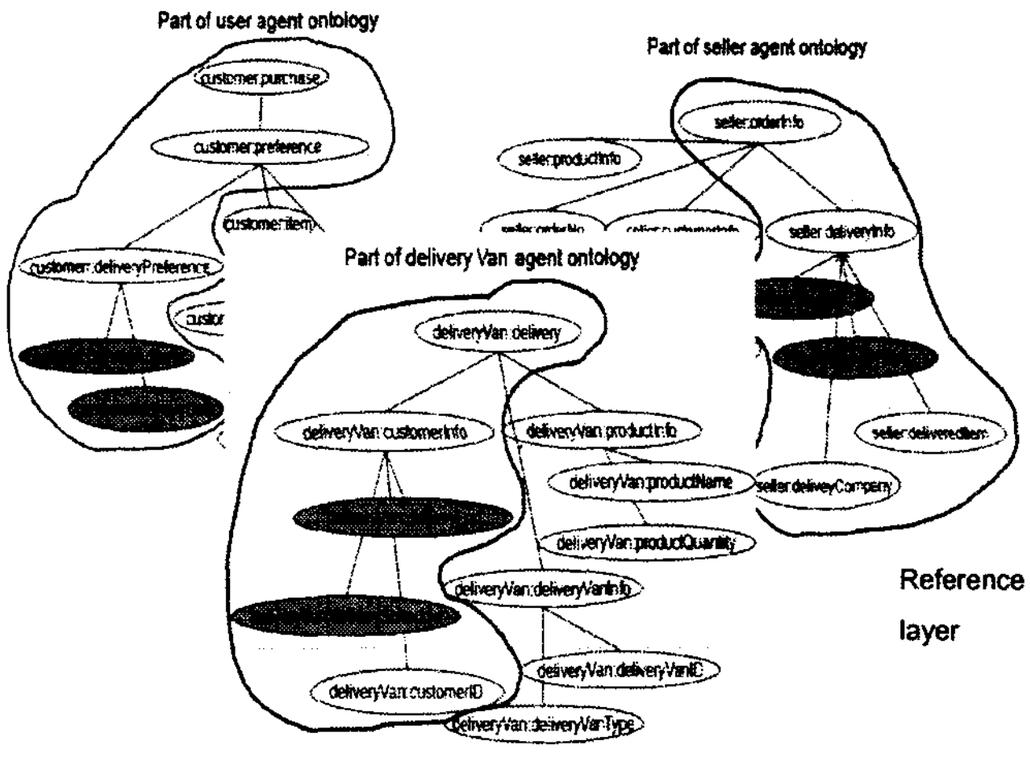

그림 3 –추출된 equivalentClass 들

그림3은 각 에이전트들로부터 문제해결에 필요한 클래스들을 포함하고 있는 부분 온톨로지 (Partial Ontology)들을 추출한 것들을 보여 주고 있다.

위 그림3으로부터 추출된 부분 온톨로지를 이용해 IOV를 구성하는 알고리즘을 표 2에 제시하였다.

표 2 – IOV 생성 알고리즘

```
Node node_Set[] //노드 배열

Function get_Every_Related_Class(Class class)

//현재 클래스와 같은 상위 클래스를 가지고 있는
클래스들의 상하위 클래스 전달
node = node -> super_Class
get_Every_Super_Class(class)
get_Every_Sub_Class(class)
class_Set[] =class
return class_Set[]
End Function

// 클래스의 모든 상위 클래스를 저장
Function get_Every_Super_Class(Class class)
  While class != root
      class = class -> super_Class
      class_Set[] =class
End Function

//node의 모든 하위노드를 저장
Function get_Every_Sub_Class(Class class)
  IF class != tail
    For each
class.get_Every_Sub_Class(class->subClass)
      class_Set[] =class
  End IF
End Function
```

위의 알고리즘을 실행한 결과 생성되는 통합 온톨로지 뷰는 그림 4에 도시하였다. 그림4는 IOV를 Root로 하여 부분 온톨로지들을 동적으로 통합한 예이다.

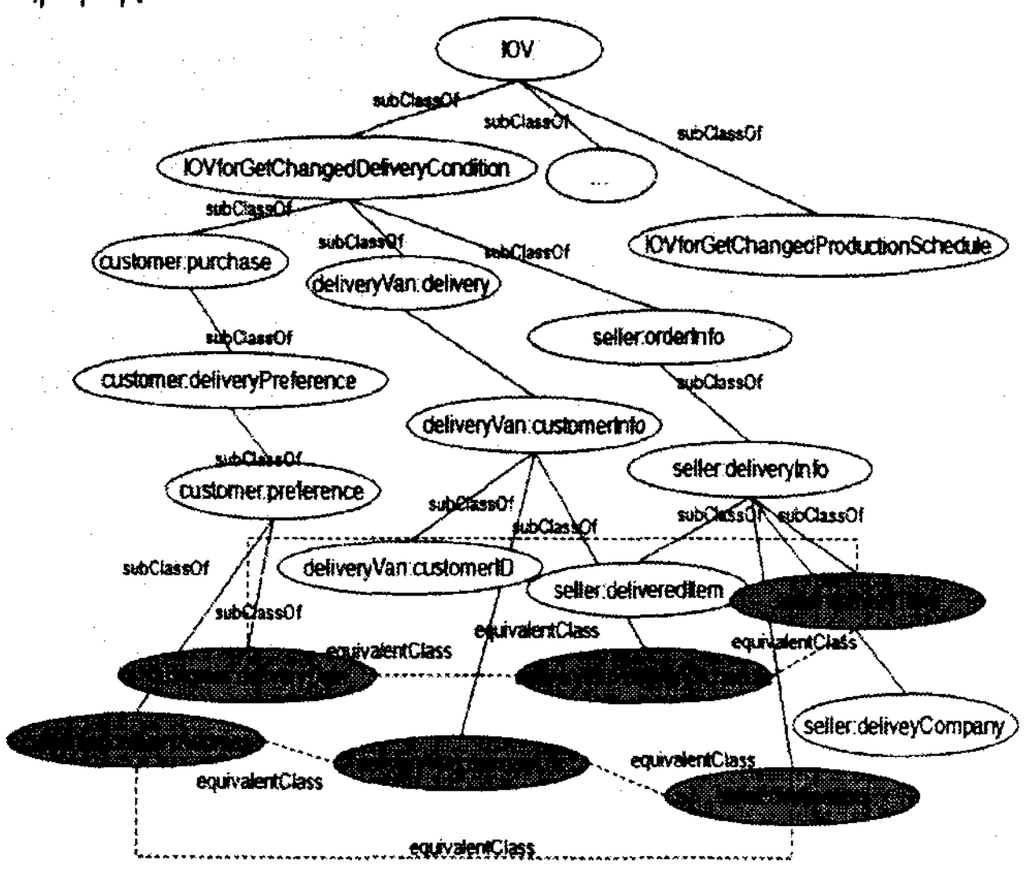

그림 4 – 통합 온톨로지 뷰 (IOV)

표 3은 온톨로지 언어인 OWL을 이용해 표현한 IOV이다.

표 3 – OWL로 표현한 IOV

```
<rdf:RDF
    xmlns:customer                                    =
http://skku.edu/e-fulfillment/user/order#
    xmlns:seller                                      =
http://www.cjmall.com/e-fulfillment/orderInfo#
    xmlns:vehicle                                     =
http://www.cjlogi.com/e-fulfillment/deliveryVan#/>

 <owl:Class rdf:ID="&customer;deliveryAddress">
     <rdfs:subClassOf
rdf:resource="&customer ;preference" />
 </owl:Class>
 <owl:Class rdf:ID="&customer; preference">
     <rdfs:subClassOf
rdf:resource="&customer ;deliveryPreference" />
 </owl:Class>
 :
  <owl:Class rdf:ID =
"&customer;deliveryAddress">
     <owl:equivalentClass rdf:resource =
"&vehicle;deliveryLoc"/>
     <owl:equivalentClass rdf:resource =
"&seller;destination"/>
 </owl:Class>

  <owl:Class rdf:ID = "&customer;deliveryDate">
     <owl:equivalentClass rdf:resource =
"&vehicle;deliverySchedule"/>
     <owl:equivalentClass rdf:resource =
"&seller;deliveryTime"/>
 </owl:Class>
```

4.2 다중 에이전트 협력 모듈에서 데이터 충돌 해결 알고리즘

다중 에이전트 협력 모듈인 MACM에서 고객의 요구사항 변화에 따라 변화된 데이터 값들간의 충돌의 해결을 위한 목적함수는 전역 목적함수와 지역 목적함수로 구분한다.

- *전역 목적함수:* 고객의 요구사항 만족을 위한 목적함수를 다중에이전트들간의 전역 목적함수로 정의한다.
- *지역 목적함수:* 지역 목적함수는 전역 목적함수를 만족하는 상황에서 데이터 변화에 따른 전파가 최소한으로 이루어지는 방향으로 정의된다.

목적함수 들간의 우선순위는 전역목적함수가 지역목적함수를 우선한다. 지역목적함수들 간에는 데이터 값 변화에 따라 그 전파가 최소한 이루어지는 방향이 우선한다.

다음 표5는 MACM에서 데이터 충돌 해결을 위한 알고리즘을 제시한다.

표5 - MACM에서 데이터 충돌 해결 알고리즘

```
Import IOV from i-OMM
Set Confliction_data_set[]=Confliction
Set Global goal
Set Local goals from multi-agent
Set Relationship (Condition) between equvalentClasses and between equivalentClass and subclass

//전역 목적함수에 따른 데이터 값 변경
Change Instance_value which satisfies with global goal

//지역 목적함수에 따른 equivalentClass들간의 데이터 값 변경
Function resove_confliction1 (Confliction Conflction)
    While confliction != null
        Change Instance_value between equivalentClasses which satisfies with local goals and global goal
End Function

//지역 목적함수에 따른 equivalentClass와 subClass간의 데이터 값 변경
Function resove_confliction2 (Confliction Conflction)
    While confliction != null
        Change instance_value between equivalentClass and subclass which satisfied with local goals and global goal.
End Function
```

결정된 데이터 값은 각 개별 에이전트에게 전달되어 Feedback을 통해 완성되도록 한다.

5. u-DSS 포털에서의 지능형 온톨로지 관리를 위한 정보처리의 절차

u-DSS포털은 앞서 제시한 바와 같이 u-DSS 에이전트, i-OMM, MACM 및 KRM으로 이루어 진다. 따라서 의사결정 지원을 위한 지능형 온톨로지 동적 통합을 위한 각각의 시스템에서의 정보처리의 순차적 절차를 살펴보면 다음과 같다.

5.1 u-DSS 에이전트의 정보 처리 절차

개별 에이전트의 스마트 기기들로부터 인식된 상황정보를 전달받아 에이전트들의 협력을 통해 문제해결을 관장하는 u-DSS에이전트의 정보처리 절차는 다음과 같다.

- ***u-DSS 에이전트에서의 정보처리 절차***

Step1: u-DSS 에이전트는 자신에게 정보를 전송하는 스마트 기기들로부터의 전달된 정보를 수신한다.

Step2: 그림 2에서와 같이 u-DSS 에이전트의 Context Identification Module (CIM)은 서비스 카탈로그 (service catalogue)와 개별 온톨로지 (individual ontology)를 사용해 사용자가 현재 처해 있는 문제 상황이 독자적으로 해결할 수 있는지 아니면 다른 에이전트들과의 협력적 해결이 필요한지를 지식공학자에 의해 기 구축된 서비스 카탈로그의 서비스 계층을 참조하여 판단한다.

Step3: CIM이 독자적인 해결이 어렵다고 판단하면 다른 에이전트와의 협력을 u-DSS 엔진에 요청한다.

5.2 i-OMM의 가상 온톨로지 뷰 (IOV)의 생성 절차

다중 에이전트들이 협력적으로 문제 해결을 하기 위해 필요한 가상의 동적 온톨로지 뷰를 생성하는 정보처리절차를 도식화하면 다음 그림 5와 같다.

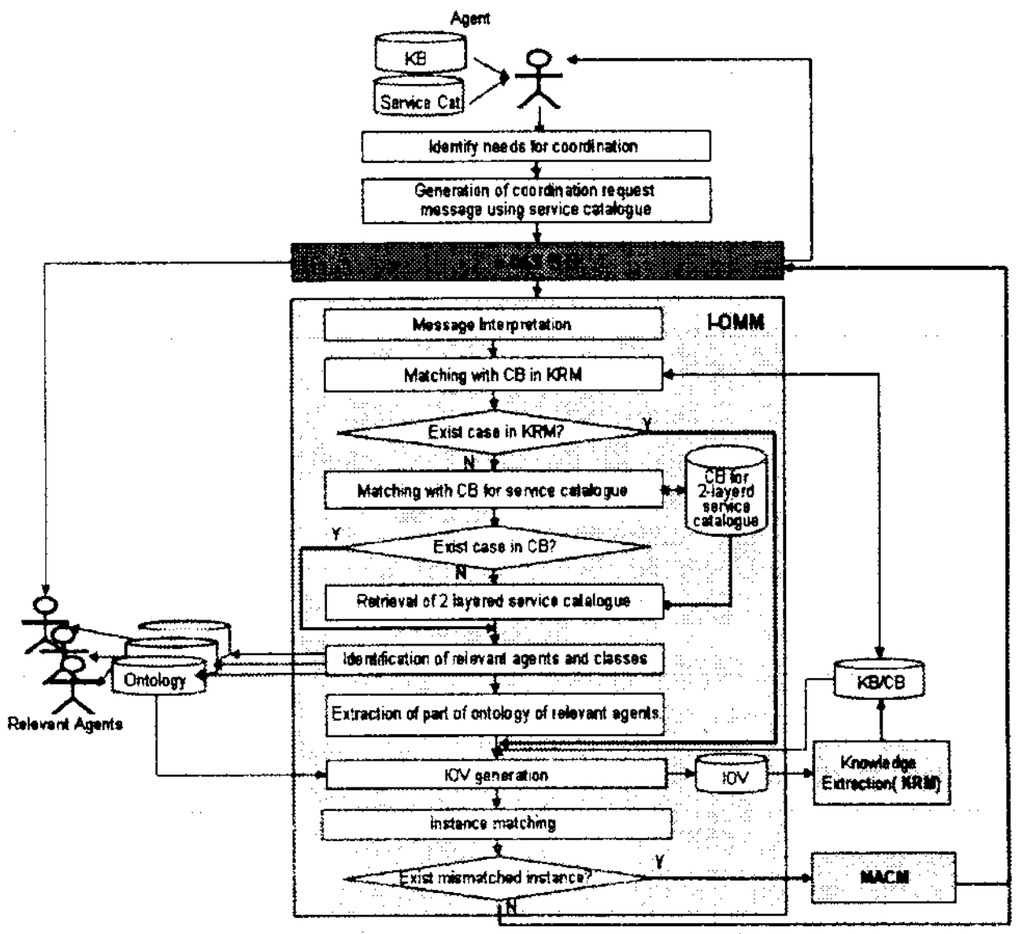

그림 5 – u-DSS엔진에서의 문제해결 절차

즉 그림5는 i-OMM을 이용하여 온톨로지들 간의 의미적 충돌 해결 위해 그림3에서 식별된 개별 온톨로지들을 기반으로 가상의 온톨로지 뷰를 생성하는 정보 처리절차를 보여준다. i-OMM에서 정보처리의 순차적 단계는 다음과 같다.

- ***i-OMM의 이질적 온톨로지 동적 통합 (IOV)을 위한 정보처리 절차***

Step1: 사례베이스로부터 유사 문제 해결을 위해 생성된 가상의 온톨로지 뷰 (IOV)가 있는지 확인한다.

Step2: 만약 사용되었던 유사 뷰가 없다면, 식별된 온톨로지 구성요소들을 통합하여 그림 4와 같은 새로운 뷰를 생성한다.

Step3: 서비스 카탈로그의 참조 계층을 이용해 클래스들의 관계, 즉 equivalenceClass, disjointWithClass 및 subClass등의 관계를 추가한다.

표6는 통합된 온톨로지를 OWL로 표현한 예이다.

표 6 – OWL로 표현된 통합 온톨로지

```
<rdf:RDF
    xmlns:customer = http://skku.edu/e-fulfillment/user/order#
    xmlns:seller = http://www.cjmall.com/e-fulfillment/orderInfo#
    xmlns:vehicle = http://www.cjlogi.com/e-fulfillment/deliveryVan#/>

 <owl:Class rdf:ID="&customer;deliveryAddress">
     <rdfs:subClassOf rdf:resource="&customer ;preference" />
 </owl:Class>
 <owl:Class rdf:ID="&customer; preference">
     <rdfs:subClassOf rdf:resource="&customer ;deliveryPreference" />
 </owl:Class>

 <owl:Class rdf:ID = "&customer;deliveryAddress">
     <owl:equivalentClass rdf:resource = "&vehicle;deliveryLoc"/>
     <owl:equivalentClass rdf:resource = "&seller;destination"/>
</owl:Class>

 <owl:Class rdf:ID = "&customer;deliveryDate">
     <owl:equivalentClass rdf:resource = "&vehicle;deliverySchedule"/>
     <owl:equivalentClass rdf:resource = "&seller;deliveryTime"/>
</owl:Class>
```

Step4: 하나의 통합 온톨로지 뷰 (IOV)를 생성한 후에는 데이터 계층(Instance level)의 데이터 값의 변화에 따른 충돌을 해결하고 협력적 문제 해결을 위해 다중 에이전트 조정 모듈 (Multi-agent coordination module, MACM)로 보내진다.

Step5: i-OMM에 의해 생성된 가상 온톨로지 뷰는 재사용을 위해 KRM의 사례베이스에 저장된다.

5.3 다중 에이전트 조정 모듈 MACM에서의 협력적 문제 해결을 위한 정보처리 절차

다중에이전트에 의한 협력적 문제 해결을 위해 온톨로지 들간의 이질적 표현에 대한 통합은 i-OMM을 통해 해결했다. 즉 Instance level의 의미적 불일치는 IOV생성에 의해 해결하고, Instance level의 고객의 요구사항 변화에 따른 데이터 값들간의 충돌은 MACM을 이용하여 해결한다.

- ***MACM의 데이터 충돌 해결을 위한 정보처리 절차***

Step1: 가상 동적 통합 뷰인 (IOV)의 데이터 계층 (Instance Level)들의 ***equivalenceClass***에서의 데이터 값의 충돌을 파악한다.

Step2: 각 에이전트들의 지역목적 함수와 인과관계 및 고객 에이전트의 전역 목적함수와 인과관계를 설정한다.

Step3: 충돌해결을 위해 목적함수들간의 우선순위를 정한다. 우선순위는 고객 에이전트의 목적함수를 우선으로 하고, 데이터 변화에 따른 전파(Propagation)가 적은 방향으로 한다.

Step4: 데이터 값 조정(Adjustment)후 u-DSS 에이전트를 통해 에이전트들과 통신(Communication) 하고 피드백을 전달 받는다.

5.4 KRM 문제해결 절차

KRM의 주요 역할은 가상의 온톨로지 뷰로부터 추후에 활용될 것으로 기대되는 정보만을 추출해, 사례로 재구성하여 사례베이스에 저장하는 것이다. 이 사례베이스는 i-OMM이 가상의 온토롤지 뷰를 구축할 때 새롭게 생성하기에 앞서 참조하게 된다. 사례의 저장과 진화는 다음 연구에서 다루기로 한다.

6. u-Fulfillment에서의 u-DSS portal 활용

본 연구의 연구영역인 u-fulfillment는 유비쿼터스 컴퓨팅환경에서 '고객 만족(customer satisfaction)'을 목표로 운영되며, 이에 더해 각각의 구매자, 판매자, 제조업자, 창고 및 물류업자 등의 각 참여자의 이익과 효용의 극대화를 동시에 추구한다. 이러한 목표를 달성하기 위해 선결되어야 할 중요 과제중의 하나가 u-fulfillment의 참여자들간의 주문, 재고 및 배송 정보 등과 같은 정보나 지식 등의 공유이다. 이질적인 구성요소들의 집합체인 u-fulfillment 망에서 정보 공유를 위한 대안중의 하나가 모든 구성요소들이 공유할 수 있는 공통의 온톨로지를 개발하는 것이다. 그러나 Gruber가 지적했듯이 공통의 온톨로지의 구축은 현실적이지도 않고 바람직하지도 않은 방법이다 [Gruber, 1993]. 따라서 이를 해결하기 위한 대안으로 본 연구에서는 앞에서 서술한 바와 같이 u-DSS 포탈을 제안했다. 본 장에서는 u-Fulfillment를 간략히 살펴보고 다음의 시나리오를 기반으로 u-Fulfillment 도메인에서의 i-OMM과 MACM의 활용을 살펴보기로 한다.

6.1 u-Fulfillment

u-fulfillment는 고객, 판매자, 제조업체 및 배송업체의 차량 등의 다양한 참여자들로 구성되며, 이들 구성요소들은 고객 만족을 위해 협력적인 문제 해결이 요구되는 분야이다. 따라서 u-DSS 포탈의 개념 구현에 적당하다 할 수 있다. 그림 1은 u-fulfillment 도메인을 추상화시키기 위한 계층구조이다.

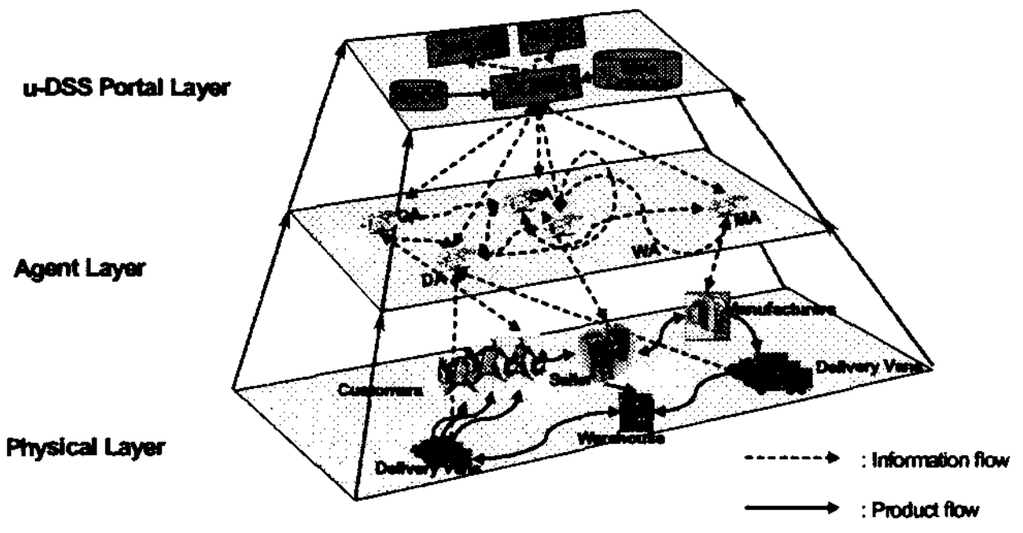

그림 1 –e-Fulfillment의 구성요소와 u-DSS포탈에서의 다중 에이전트의 도식화

물리적인 계층에 존재하는 u-fulfillment

구성요소들은 각각 자신들의 의사결정을 지원해 주는 소프트웨어 에이전트를 가지고 있다고 가정한다. 에이전트 계층에 존재하는 각각의 에이전트들은 물리적 계층에 있는 자신의 구성요소로부터 정보를 전달받아 자신의 온톨로지와 추론엔진을 이용해 문제 해결을 하거나, 필요하면 다른 에이전트들과 정보를 교환하면서 문제를 해결한다. 이 과정에서 에이전트들간에 조정이 필요하게 되면 u-DSS 엔진계층에 조정을 요청하게 된다. u-DSS 엔진이 에이전트들간에 발생한 온톨로지 충돌(confliction)을 조정하기 위해 IOV를 생성한다. 즉 연관되어 있는 에이전트들이 가지고 있는 개별 온톨로지의 구성요소들 중 특정 문제해결에 필요한 요소만을 간추린 - 이 과정에서 문법적·의미적 충돌이 해결됨 - 하나의 새로운 온톨로지를 동적으로 통합해 조정을 수행하게 된다. 예를 들어, 고객이 상품을 주문한 후, 판매자가 물품을 고객에게 배달하는 과정에서 여러 가지 예측하지 못한 변수의 등장으로 문제가 발생할 수 있다. 고객이 배달 일자나 장소를 변경하거나, 판매자가 갑작스런 재고 소진으로 배송을 지연할 수 있고, 생산자의 내부적인 문제로 생산일정에 차질이 발생할 수도 있다. 이러한 문제를 해결하기 위해서, **u-fulfillment** 구성요소들의 에이전트는 변화된 상황을 인지해야 하며, 인지된 상황으로 인해 발생된 에이전트들간의 온톨로지 충돌(confliction)을 해결하기 위해 IOV를 생성하게 된다. 또한 이 과정에서 변화된 데이터에 의해 다중 에이전트들간의 충돌이 발생할 수 있으며 이를 해결하는 시스템이 MACM이다. 데이터간의 충돌은 IOV에 의해 생성된 동적 통합 온톨로지를 중심으로 다중 에이전트들 간의 Instance level들의 데이터 값간의 충돌을 의미한다. 또한 Instance level에서 인과관계로 연계된 Instance와의 데이터 값 변화에 따른 전파(Propagation)이 존재한다. 따라서 변화된 데이터 값에 따른 다중 에이전트 들간의 이익최대화와 데이터 변환과 이로 인한 Propagation의 최소화를 MACM이 지원한다.

6.2 u-DSS 포탈의 활용 예제

고객이 배송 목적지를 서울에서 수원으로 일자를 2007년 4월9일에서 2007년4월12일로 변경하였다. 위와 같은 고객의 상황변화는 고객 에이전트에 탑재되어 있는 CIM이 인지한 후, 자체적으로 해결이 가능한 문제인지 또는 다른 에이전트와의 조정이 필요한 문제인지를 판단한다. 다른 에이전트들과 조정이 필요하면 u-DSS 엔진에 요청하여 다른 에이전트들과의 조정을 하게 된다.

다음 그림5는 에이전트들간의 조정을 u-DSS 엔진을 통해 해결해야 하는 이유를 표현하고 있다. 예를 들어, 그림 5에서와 같이 u-Fulfillment에 관련되어 있는 쇼핑몰과 배송업체가 상호 협력을 해야 하는 경우, 이 둘간의 온톨로지 사이에 존재하는 문법적 그리고 의미적 불일치는 문제 해결을 위한 직접적인 협력을 어렵게 한다.

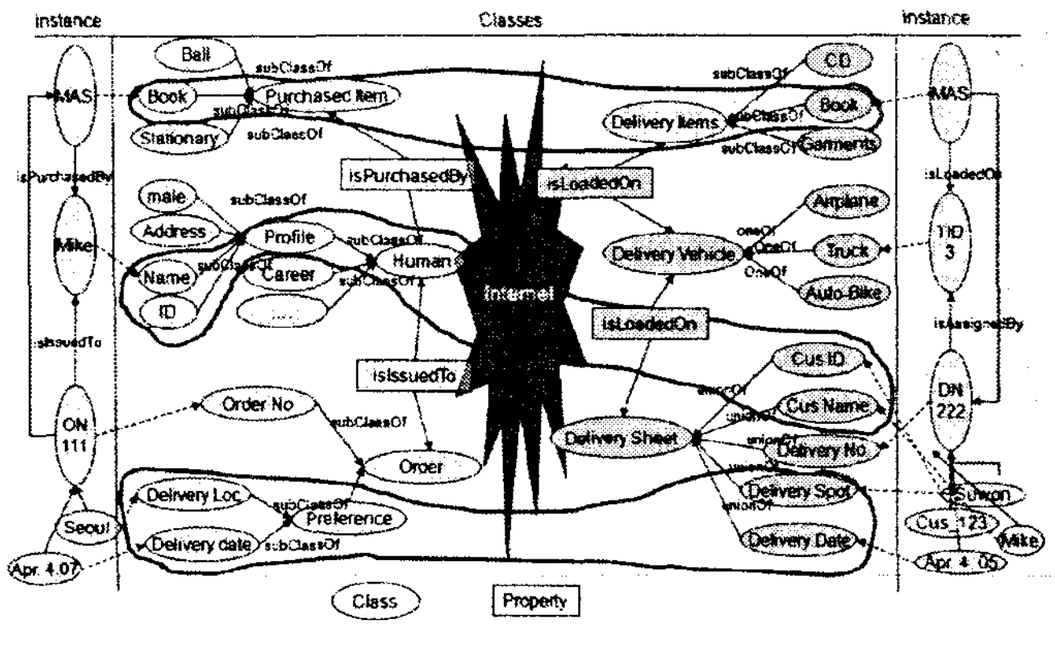

그림 5 –에이전트들간에 발생할 수 있는 의미적 불일치 예

따라서 그림5에서 제시된 상황을 중심으로 다중 에이전트 들, 즉 고객에이전트, 판매자 에이전트, 배송차량 에이전트들간의 온톨로지의 문법적, 의미적 충돌의 해결과 고객 에이전트의 요청에 의한 데이터 변화로 야기된 에이전트들간의 데이터 충돌을 해결하는 간단한 예제를 4장의 설명을 위해 제시된 예와 그림 들을 예제로 하여 살펴보기로 한다.

다른 정보자원으로부터 유사한 개념들과 관계들에 사용된 매핑 방법은 Equivalence Relation[23]에 의해 각각의 관계가 정의된다. 예를 들어, 고객에이전트의 'deliveryAddress'와 'deliveryDate' 클래스들은 배송 에이전트의 'deliveryLoc'와 'deliverySchedule' 클래스들과 판매자 에이전트의 'destination'와 'deliveryTime'과 ***equivalentClass***로 문제 해결을 위한 연결이 요구된다. 또한 고객에이전트의 'deliveryAddress'의 현재 데이터(Instance) 값은 '수원'이고, 배송차량(van) 에이전트의 클래스 'deliveryLoc'과 판매자 에이전트의 클래스 'destination'이 그들 각각의 데이터 (Instance) 값으로 '서울'을 가지고 있어 데이터 충돌이 방생한다. 'deliveryDate'와 'deliveryTime' 및 'delivery Schedule'간에도 데이터 충돌을 확인할 필요가 있다. 예와 같이 다중 에이전트의 협력을 통한 문제해결에 있어 각 에이전트들이 가지고 있는 이질적 온톨로지와 데이터 값간의 충돌을 해결하기 위한 처리절차에 따라 예시하면 다음과 같다.

U-DSS 에이전트

Step1: 스마트 기기를 통해 고객에이전트로부터 고객의 변경된 요구사항 수신한다.

배송위치: 서울 → 수원

배송일자: 2007/04/09 → 2007/04/12

Step2: CIM을 이용하여 문제인식 후 독자적인 문제해결이 가능한지 아니면 협력적 문제 해결이 가능한지 판단한다.

표1의 서비스 카탈로그의 서비스계층에 의하면

고객 에이전트의 위치 및 일자에 해당하는 데이터 계층(Instance Level)은 각각 'deliveryAddress'와 'deliveryDate'이다. 이들은 서비스 카탈로그의 참조계층에 의하면 배송 에이전트인 'DeliveryVanAgent'와 판매자 에이전트인 'Delivery SellerAgent'와 협력적 문제 해결이 요구됨을 확인할 수 있다.

Step3: Step2에서 고객의 요구사항의 변경 요청은 배송 에이전트인 'DeliveryVanAgent'와 판매자 에이전트인 'SellerAgent'와 협력적 문제해결이 요구됨이 확인되면 u-DSS 엔진에 문제해결을 요청한다.

U-DSS 엔진

i-OMM

Step1: UDSS 에이전트에서의 CIM의 결과를 바탕으로 문제 해결에 필요한 IOV가 있는지 사례베이스를 검색한다.

Step2: 가상 온톨로지 뷰 생성

Step2.1 사례베이스에서 유사 뷰 탐색

유사 뷰를 발견하면 이를 재 사용한다. 추후 연구에서 다루기로 한다.

Step2.2: 유사뷰가 없다면 IOV 생성

각 에이전트들로부터 부분 온톨로지 추출

그림3에서와 같이 각 구성요소 에이전트들의 온톨로지들로부터 고객의 요구사항에 해당하는 데이터 계층 (Instance Level)을 가지고 있는 부분 온톨로지를 추출하여 그림4와 같이 새로운 뷰 (IOV)를 생성한다.

Step3: 클래스들간의 관계정의

equivalentClass의 연결

고객 에이전트인 'CustomerAgent'의 'deliveryAddress'는 판매자 에이전트인 'SellerAgent'의 'destination' 및 배송에이전트인 'DeliveryVanAgent'의 'deliveryLoc'과 ***equivalentClass***로 연결된다. 고객 에이전트의 'deliveryDate'는 판매자 에이전트인 'SellerAgent'의 'deliveryTime'과 배송 에이전트인 'DeliveryVanAgent'의 'deliverySchedule'과 ***equivalentClass***로 연결된다. ***equivalentClass***의 하위 데이터 계층에 연결된 클래스들은 ***subClass***로 정의된다.

예)

equivalentClass

ClassSet_equivalent1

Van.deliveryLoc
Customer.deliveryAddress
Seller.destination

ClassSet_equivalent2

Van.deliverySchedule
Customer.deliveryDate
Seller.deliveryTime

disjointWithClass

null

subClass

Seller.deliveryTime
Seller.deliveryCompanuy

Step4: 각 에이전트들로부터 온톨로지의 의미적 문법적 충돌 해결을 위해 생성된 IOV를 이용하여 IOV내의 데이터 계층 (Instance Level)의 데이터 충돌을 해결하기 위해 MACM을 호출한다.

Step5: 새롭게 생성된 IOV는 KRM에 의해 사례베이스에 저장된다.

MACM

Step1: 데이터 계층 (Instance Level)의 충돌 해결

고객에이전트의 요구사항 중 위치가 '서울'에서 '수원'으로 변경되었으므로 배송차량과 판매자 에이전트들의 데이터 값 확인이 필요하다.

다음과 같이 배송차량 에이전트와 판매자 에이전트의 데이터 값이 여전히 '서울'로 되어 있으므로 ***equivalentClass***간의 데이터 충돌이 있음을 확인한다.

Van.deliveryLoc(서울)
<>Customer.deliveryAddress(수원)
<>Seller.destination(서울)

또한 배달일자에 해당하는 ***equivalentClass***들간에도 다음과 같은 데이터 충돌이 있음을 확인한다.

Van.deliverySchedule(2007/04/09)
<>Customer.deliveryDate(2007/04/12)
<> Seller.deliveryTime(2007/04/09)

Step2: 충돌 해결을 위한 전역목적함수와 지역목적함수설정

u-fulfillment에서의 전역목적함수는 고객의 요구사항 만족이다. 따라서 고객 에이전트의 데이터 변경 요청에 의해 배송차량과 판매자 에이전트의 ***equivalentClass***에 해당하는 데이터 값을 변경한다.

전역목적함수:

Customer.deliveryAddress(수원)
Customer.deliveryDate(2007/04/12)

인과관계:

Customer.deliveryAddress(수원)
= Van.deliveryLoc // 목적함수와 대등관계
= Seller.destination // 목적함수와 대등관계

Customer.deliveryDate(2007/04/12)
= Van.deliverySchedule // 목적함수와 대등관계
= Seller.deliveryTime // 목적함수와 대등관계

지역목적함수는 전역목적함수에 의해 데이터 값 변경에 따른 ***subClass***의 데이터 값 변경이 요구될 때 데이터 값 변경의 전파가 적은 방향으로 변경할 데이터를 탐색한다.

지역목적함수:

'Seller.deliveryCompany'가 'Seller.deliveryTime'의 ***subClass***로서 'Seller.deliveryTime'의 값 변경에 따라 'Seller.deliveryCompany' 데이터 값의 변경이 요청될 수 있다. 본 예제에서는 subClass가 하나 밖에 존재하지 않아 데이터 값 변화에 따른 전파(Propagation)의 대상이 없어 'Seller.deliveryCompany'의 데이터 값을 변경하기로 한다.

Step3: 데이터 계층 (Instance Level)에서 데이터 값 변경에 따른 충돌 발생시 전역목적함수가 지역목적함수를 우선하는 우선 순위에 따라 충돌을 해결한다. 예를 들어 'Seller.deliveryTime'은 전역목적함수와 연관되고 'Seller.deliveryCompany'는 지역목적함수와 연관되므로 'Seller.deliveryTime'의 변화에 따라 'Seller.deliveryCompany'의 데이터 값이 자동으로 변경된다.

Step4: 데이터 값 조정 후 u-DSS 에이전트를 통해 각각의 협력 에이전트들인 배송차량 에이전트, 고객 에이전트, 판매자 에이전트들에게 변경된 데이터를 전송하고 이에 대한 피드백을 받는다.

KRM

i-OMM에 생성된 그림4의 새로운 IOV를 사례베이스에 저장하여 다음의 유사한 사례에 사용하도록 한다.

7. 결론 및 향후과제

본 연구에서 제시한 u-DSS 포탈은 유비쿼터스 컴퓨팅환경에서 다중에이전트간의 협력적 문제해결 중의 의사결정지원을 위한 시스템이다. 특히 이번 연구에서는 i-OMM을 통해 다중에이전트의 협력적 문제 해결에서 발생될 수 있는 이질적 온톨로지의 동적 통합 뷰(IOV)의 생성을 이용한 문제 해결방법과 MACM을 이용한 데이터 간의 충돌 해결 방안을 제시하였다. KRM을 이용하여 사례를 저장된 통합 뷰(IOV)의 재활용과 진화는 다음 연구에서 다루기로 한다.

본 연구에서 제시된 가상 온톨로지 통합 뷰는 다중 에이전트들이 협력적 문제해결을 할 때 통상의 통합 뷰를 요구하는 데 비하여, 제시된 뷰는 참여 에이전트와 상황에 따라 필요한 온톨로지들 간의 동적인 통합 뷰의 생성이 가능하여 에이전트들 간의 정보처리 부담을 줄이고, 다중 참여자들을 기반으로 하는 의사결정의 효율성을 증대시킨다.

본 연구에서는 u-DSS 포탈의 연구 영역으로 **u-Fulfillment**를 선택하였다. **u-Fulfillment**는 다양한 참여자로 구성되어 다중 에이전트들간의 협력을 필요로 하고, 고객의 요구사항이 변화될 때 마다 참여 에이전트들의 상황인식에 따른 즉각적 대응이 요구된다는 특징이 있다. 따라서 **u-Fulfillment**를 대상으로 하여 간단한 예를 제시하였고, 본 연구에서 제시한 u-DSS 포탈의 성능 및 효율성의 평가가 요구된다.

8. 참고문헌

[1] Banavar, G. and Bernstein, A., "Mobile-Agent Coordination Models for Internet Applications," Computer, Vol. 33, No. 2 (2000), 82-89.

[2] BenMoussa, C., "Workers on the Move: New Opportunities through Mobile Commerce," The Stockholm Mobility Roundtable, (May, 2003), 22-23.

[3] Boudriga, N. and Obaidat, M.S., "Intelligent Agents on the Web: A Review," IEEE Computing Science & Engineering, Vo. 6, No. 4 (July/August, 2004), 35-41.

[4] Calvanese, D., De Giacomo, G., and Lenzerini, M., "A framework for ontology integration," In Proceedings of the 1st Internationally Semantic Web Working Symposium (SWWS), Stanford, CA, USA, (August, 2001).

[5] Doan, A., Madhavan, J., Domingos, P., and Halevy, A., "Learning to Map between Ontologies on the Semantic Web," WWW2002, Honolulu, Hawaii, USA, (May, 2002).

[6] Fernandez-Breis, J. and Mart´ınez-B´ejar, R., "A cooperative framework for integrating ontologies," International Journal of Human-Computer Studies, Vol. 56, No. 6 (2002), 665-720.

[7] Gordon, B.D., "Anytime/Anyplace Computing and the future of knowledge work," Communication of the ACM, Vol. 45, No. 12 (2002), 67-73.

[8] Gruber, T. R., "Toward principles for the design of ontologies used for knowledge sharing," International Workshop on Formal Ontology in Conceptual Analysis and Knowledge Representation, Padova, Italy, (1993).

[9] Hendler, J., "Agents and the Semantic Web," IEEE Intelligent System, (March/April, 2001), 30-37.

[10] Keen, P. and Mackintosh, R., "The Freedom Economy: Gaining the M-commerce Edge in the Era of the Wireless Internet", Osborne/McGraw-Hill, Berkeley, CA, (2001).

[11] Klein, M., "Combining and Relating Ontologies: An Analysis of Problems and Solutions, Workshop on Ontologies and Information Sharing," Seventeenth International Joint Conference on Artificial Intelligence (IJCAI), Seattle, Washington, USA, (August, 2001).

[12] Lottaz, C., Smith, I.F.C., Robert-Nicoud, Y., and Faltings, B. V., "Constraint-based support for negotiation in collaborative design," Artificial Intelligence in Engineering, Vol. 14, No. 3 (2000), 261-280.

[13] Luo, X., Zhang, C., and Leung, H.F., "Information sharing between heterogeneous uncertain reasoning models in a multi-agent environment: A case study," International Journal of Approximate Reasoning, Vol. 27, No. 1 (2001), 27-59.

[14] Lyytinen, K. and Yoo, Y., "Research Commentary: The Next Wave of Nomadic Computing," Information Systems Research, Vol. 13, No. 4 (2002), 377-388.

[15] McGuinness, D.L., Fikes, R., Rice, J., and Wilder, S., "An Environment for Merging and Testing Large Ontologies," Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000). Breckenridge, Colorado. (April, 2000).

[16] McMullen, P.R., "An ant colony optimization approach to addressing a JIT sequencing problem with multiple objectives," Artificial Intelligence in Engineering, Vol. 15, No. 3 (2001), 309-317.

[17] Mitra, P., Wiederhold, G., and Kersten, M.L., "A Graph-Oriented Model for Articulation of Ontology Interdependencies," Proceedings of the 7th International Conference on Extending Database Technology: Advances in Database Technology, Konstanz, Germany, (2000).

[18] Noy, N.F. and Musen, M., "PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment," In Proceedings of the 17th National Conference on Artificial Intelligence (AAAI'00), Austin, TX, USA, (July, 2000).

[19] Noy, N.F. and Musen, M.A., "Evaluating Ontology-Mapping Tools: Requirements and Experience," In Workshop on Evaluation of Ontology Tools at EKAW'02 (EON2002), Sigüenza, Spain, (October, 2002).

[20] Pazienza, M.T., Stellato, A., Vindigni, M., Valarakos, A., and Karkaletsis, V., "Ontology integration in a multilingual e-retail system," HCI International 2003, Crete, Greece, (2003).

[21] Pinto, H.S. and Martins, J.P., "A Methodology for Ontology Integration," First International Conference on Knowledge Capture (K-CAP 2001), Victoria, British Columbia, (October, 2001).
[22] Ram, S. and Park, J.. "Semantic Conflict Resolution Ontology (SCROL): An Ontology for Detecting and Resolving Data and Schema-Level Semantic Conflicts," IEEE Trans. on Knowledge and Data Engineering, Vol. 16, No. 2 (2004), 189-202.
[23] Shim, J.P., Warkentin, M., Courtney, J.F., Power, D.J., Sharda, R., and Carlsson, C., "Past, present, and future of decision support technology," Decision Support Systems, Vol. 33, No. 2 (2002), 111-126.
[24] Ulieru, M., Norrie, D., Kremer, R., and Shen, W., "A multi-resolution collaborative architecture for web-centric global manufacturing," Information Sciences, Vol. 127(1-2) (2000), 3-21.
[25] D.J. Wu, "Software agents for knowledge management: Coordination in multi-agent supply chains and auctions," Expert Systems with Applications, Vol. 20, No. 1 (2001), 51-64.

Acknowledgement

이 논문은 2006 년도 성균관대학교 Post-Doc. 연수지원에 의하여 연구되었음.