Ad Hoc한 대화 정책을 지원하는 멀티 에이전트 플랫폼에 관한 연구

안 형 준†

요 약

멀티 에이전트 시스템은 분산 환경에서 독립된 소프트웨어 개체들의 지능형 협력 작업을 지원하기 위해 개발되었으며 많은 응용 환경에서 사용되어지고 있다. 멀티 에이전트 시스템의 협력을 위해서는 에이전트 간에 미리 협의된 형태의 프로토콜인 대화 정책(Conversation Policy; Interaction Protocol)이 사용된다. 현재의 동적인 인터넷 전자시장 환경에서는 거래의 형태가 변화함에 따라 대화 정책이 자주 바뀔 수 있으며 따라서 Ad Hoc한 형태의 대화 정책의 중요성이 커지고 있다. 기존의 에이전트 플랫폼은 몇 개의 표준 혹은 미리 정해진 대화 정책만을 허용함으로써 Ad Hoc한 대화 정책에 대해서는 에이전트 시스템을 일부 새로 구현해주어야 하는 번거로움과 비효율성이 존재한다. 본 연구에서는 그러한 Ad Hoc한 대화 정책을 지원하는 에이전트 플랫폼을 설계하며 그 프로토타입 시스템을 제시한다. 제시된 시스템은 교환 및 해석을 위한 대화 정책 모델, 새로운 대화 정책을 처리하기 위한 메타 대화 처리 부분, 그리고 대화 정책을 런타임시에 교환하고 해석하여 실제 에이전트들이 적응성 있는 상거래 및 협력을 할 수 있도록 하는 메커니즘을 포함한다.

A Multi-Agent Platform Capable of Handling Ad Hoc Conversation Policies

Hyung Jun Ahn†

ABSTRACT

Multi-agent systems have been developed for supporting intelligent collaboration of distributed and independent software entities and are being widely used for various applications. For the collaboration among agents, conversation policies (or interaction protocols) mutually agreed by agents are used. In today's dynamic electronic market environment, there can be frequent changes in conversation policies induced by the changes in transaction methods in the market, and thus, the importance of ad hoc conversation policies is increasing. In existing agent platforms, they allow the use of only several standard or fixed conversation policies, which requires inevitable re implementation for ad hoc conversation policies and leads to inefficiency and intricacy. This paper designs an agent platform that supports ad hoc conversation policies and presents the prototype implementation. The suggested system includes an exchangeable and interpretable conversation policy model, a meta conversation procedure for exchanging new conversation policies, and a mechanism for performing actual transactions with exchanged conversation policies in run time in an adaptive way.

키워드 : 멀티 에이전트 시스템(Multi Agent System), 대화 정책(Conversation Policy), 전자상거래(Interaction Protocol), 변화 대응(Change Adaptation)

1. 서 론

1980년대 후반에 분산 인공지능 연구의 연장선상에서 개발되어진 멀티 에이전트 시스템은 분산성, 자율성, 지능성, 협력성 등의 여러 장점을 바탕으로 현재까지 널리 연구 및 활용되어지고 있다[18]. 분산성이란 네트워 환경에서 지리적으로 분산된 에이전트들간의 협력이 자연스럽게 지원된다는 점을 의미하며, 자율성이란 기존의 소프트웨어들과 달리 에이전트 각각의 개체가 일정 수준의 독립적인 의사결정에 따라 인간 사용자를 대신하여 작업을 수행함을 의미한다. 지능성은 분산 인공 지능 연구의 연장선상에서 에이전트 시스템들이 지식 기반 추론 등의 기법을 사용하여 복잡한 문제를 해결함을 의미하며, 협력성이란 앞서 언급된 '자율성'과 '분산성'에 기반을 두고 에이전트들이 협력적으로 복잡한 문제를 해결할 수 있음을 의미한다[8, 18].

위와 같은 여러 장점들로 인해 멀티 에이전트 시스템은 다양한 분야에 활용되어져 왔다. 예로써, 에이전트에 기반한 온라인 여행 시스템, 공급사슬 관리를 위한 시스템, 자동 거래를 위한 경매 시스템 등 여러 응용프로그램들이 개발되어 왔다[1, 3, 4, 7, 9]. 또한 에이전트에 관한 표준 기구인 FIPA

† 준 회 원 : KAIST 테크노경영대학원 경영공학
논문접수 : 2004년 2월 18일, 심사완료 : 2004년 7월 3일

에서는 멀티 에이전트 시스템을 구성하는 각 요소에 대해 활발히 표준을 수립하고 이를 확산시키기 위한 노력을 기울이고 있다[13].

에이전트들이 전자 시장에서 자동화된 거래를 수행하거나 기타 협력을 하기 위해서는 보통 여러 차례의 형식화된 메시지 교환을 수행하게 되는데 이때 그러한 메시지 교환 방식을 규정하기 위해 대화 정책(Conversation Policy 혹은 Interaction Protocol)이 사용된다[12, 21]. 그러나, 정보 기술이 급속히 발전하고 새로운 비즈니스 모델이 등장하며 공급자와 구매자간의 관계가 기존의 시장에 비해 훨씬 동적인 형태를 띄는 전자시장에서는 바뀌는 거래의 형태에 따라 대화 정책에도 잦은 변화가 수반되게 되며 이는 멀티 에이전트 시스템의 적응성 측면에 큰 문제를 일으켜서 멀티 에이전트 시스템 활용의 장점을 크게 감쇄 시키게 된다[4, 15, 23].

따라서 본 논문에서는 그러한 동적인 환경에서 자주 등장할 수 있는 Ad Hoc한 형태의 대화 정책에 적응할 수 있는 멀티 에이전트 플랫폼을 설계하고 그 프로토타입 시스템을 구현함으로써 실용 가능성을 가늠해 본다. 본 논문의 OCAgent(Open Conversation Agent)는 다음과 같이 구성된다. 첫째, 에이전트들이 Ad Hoc한 대화 정책을 교환하고 해석할 수 있도록 더욱 구체적으로 명세가 가능한 대화 정책 모델이 제시된다. 둘째, 그러한 Ad Hoc한 대화 정책을 맞닥뜨렸을 때에 이를 런타임에 교환할 수 있는 절차가 제공된다. 셋째, 교환된 대화 정책을 가지고 실제로 상거래를 수행할 수 있도록 해 주는 메커니즘이 제시된다.

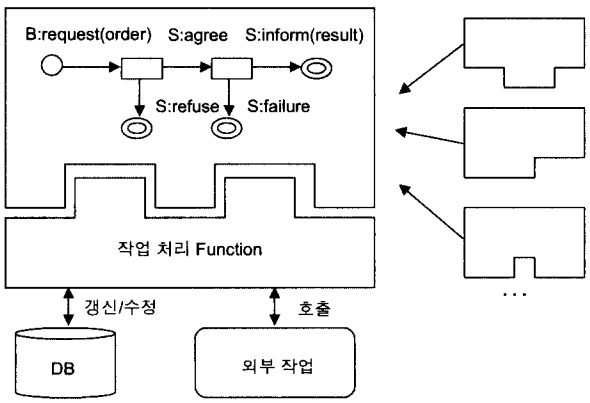
본 논문은 다음과 같이 구성되어진다. 2장에서는 관련 연구가 소개된다. 3장에서는 OCAgent의 설계가 제시된다. 4장에서는 OCAgent의 프로토타입 구현에 관한 사항이 소개된다. 5장에서는 토론과 결론을 제시하며 논문을 끝맺는다.

2. 관련 연구

2.1 동적 환경에서 고정된 대화 정책의 문제점

전자시장에서 상거래를 위해 사용되는 멀티 에이전트 시스템은 상거래의 방식, 교환되는 정보의 종류, 지불 방식 등 다양한 요소에 의해 규정되어지는 일련의 메시지 교환 방식을 고정하여 사용하게 된다. 이러한 메시지 교환 방식은 대화 정책에 반영되며, 따라서 그러한 요소들에 변화가 있을 경우에는 대화 정책 또한 변화를 수반하게 된다. 따라서, 전자시장에서 자동화된 거래를 "거래 및 관련 행위를 위한 협력적 정보 교환의 프로세스"라고 정의할 경우 멀티 에이전트 시스템들이 당면하게 되는 '거래의 변화'는 바로 '대화 정책의 변화'와 동일한 의미를 갖게 된다[4, 15].

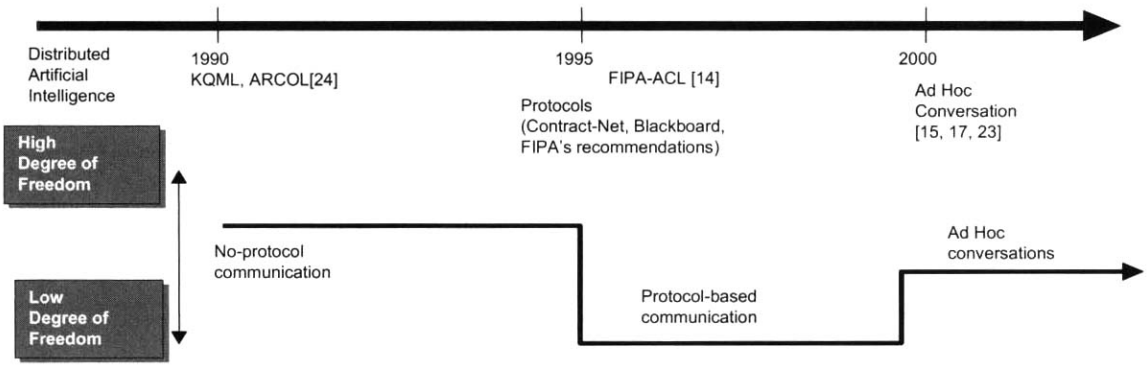
이러한 문제를 예를 들어 살펴보면 (그림 1)과 같다. 어떠한 에이전트 시스템이 그림 윗부분에 스테이트 트랜지션 다이어그램 형태로 표시된 대화 정책에 따라 상대방 에이전트와 협력을 한다고 가정하면, 이때 대화를 통해 실제 업무를 수행하는 작업처리 function은 미리 정해진 메시지 교환 패턴을 가정하여 구현되어진다. 즉, 어떠한 메시지가 어떤 시점에 도착할 지, 작업 처리 결과로 어떠한 메시지를 내보내야 하는 지가 사전에 고정되어 있으므로 작업 처리 function과 대화 정책 처리 부분은 밀접하게 결합되어 구현되어진다. 따라서, 그림의 우측과 같이 거래 방식의 변화에 따라 대화 정책에 다양한 변화가 존재할 시에는 고정된 작업 처리 function이 그러한 변화들에 대응할 수 없게 된다. 따라서 크고 작은 변화가 빈번한 전자시장 환경에서는 이러한 멀티에이전트 시스템은 적응성 측면에서 큰 한계를 갖게 된다.



(그림 1) 고정된 대화 정책의 문제점

2.2 관련 연구

(그림 2)와 같이, 멀티 에이전트 시스템의 연구는 1980년대 후반의 분산인공지능 연구의 연장선상에서 시작되었으며 1990년대 초반에는 KQML이나 ARCOL과 같은 에이전트 대화 언어가 등장하였다[19, 24]. 이 시점에서는 에이전트의 대화와 협력에 있어서 프로토콜이나 대화 정책의 사용이 필수가 아니었으며, 대신 멀티 에이전트 시스템들은 자유로운 지식 교환을 통해 지능적인 결과를 낼 수 있으리라 기대되어졌다. 그러나 대화 정책 등을 사용하지 않는 자유로운 대화 형태는 실제 복잡한 거래나 협력의 형태에 성공적인 결과를 내지 못하였고 따라서 1990년대 중반 이후에는 FIPA가 제안하는 인터액션 프로토콜(컨트랙트 넷, 경매 등)과 같은 다양한 대화 정책들이 복잡한 상거래 등을 위해 사용되기 시작하였다[13]. 두 가지 형식의 정책을 비교해 보면, 전자는 에이전트 시스템들에게 높은 정도의 대화 자유도를 준 반면 후자는 에이전트 시스템들의 대화의 자유도를 엄격히 제한하는 방식을 취하고 있다. 그러나 앞서 2.1절에서 설명된 바와 같은 문제점들로 인해 2000년 무렵부터는 Ad Hoc한 형태의 대화를 지원하는 에이전트 대화모형에 관한 연구들이 등장하고 있다[10, 15, 17]. 이런 종류의 새로운 연구들은 앞 선 두 가지 형태의 대화 모형의 장단점을 보완하는 절충적인 형태를 띠고 있다.



(그림 2) 멀티 에이전트 시스템에서 대화 정책 관련 변화

<표 1>은 그러한 Ad Hoc한 대화 정책과 관련된 연구들을 간단히 요약한 것이다. 첫 번째 연구는 에이전트 시스템들이 대화 정책을 팩키지로 구성하여 이를 동적으로 교환하게 하였고 더불어 대화 정책의 상태 전이 규칙도 동적으로 교환하게 하였다. 두 번째 연구는 XML을 사용하여 대화 정책을 정의하고 교환하도록 하였으며, 에이전트들이 메시지를 선택하는 데 도움을 주는 규칙을 제시하고 있다. 세 번째 연구는 중앙 전략 에이전트를 사용하여 대응 에이전트들의 변화를 중앙 전략 에이전트가 담당하도록 하여 대응 에이전트와 협력하는 다른 에이전트들은 변화에 무관하게 협력할 수 있도록 하는 방식을 제안하였다. 네 번째 연구는 페트리넷에 기반한 대화 정책 모형을 제시하고 이를 에이전트들이 교환할 수 있게 하였으며, 페트리넷의 장점인 분석 기능을 활용하여 대화 정책의 오류를 점검할 수 있게 하였다. 이러한 네 가지 연구들은 각각 장점과 시사점이 존재하지만 몇 가지 한계점을 갖고 있다. 첫째, 앞서 2.1절에서 제시된 바와 같이 고정된 작업 처리 function이 대화 처리 부분과 어떻게 분리될 수 있는지가 불명확하다. 둘째, 미리 알 수 없는 종류의 메시지가 수신 혹은 발신될 때 이를 어떻게 처리할 수 있는지 명시되지 않았다. 셋째, 세 번째 연구의 경우 에이전트 시스템들의 대화가 항상 중앙 에이전트를 통해 이루어지도록 강제함으로써 에이전트들의 대화의 자율성을 크게 떨어뜨리며 또한 중앙 에이전트의 수동 갱신의 부담을 주게 된다. 따라서 전반적으로 Ad Hoc한 대화 정책을 지원하기 위해서는 단순히 대화 정책을 교환할 수 있게 하는 수준 뿐 아니라, 교환 후에 고정된 작업 처리 function과 변화하는 대화 정책 처리 부분을 어떻게 분리, 결합 시킬 수 있는지 명확히 정리될 필요가 있다.

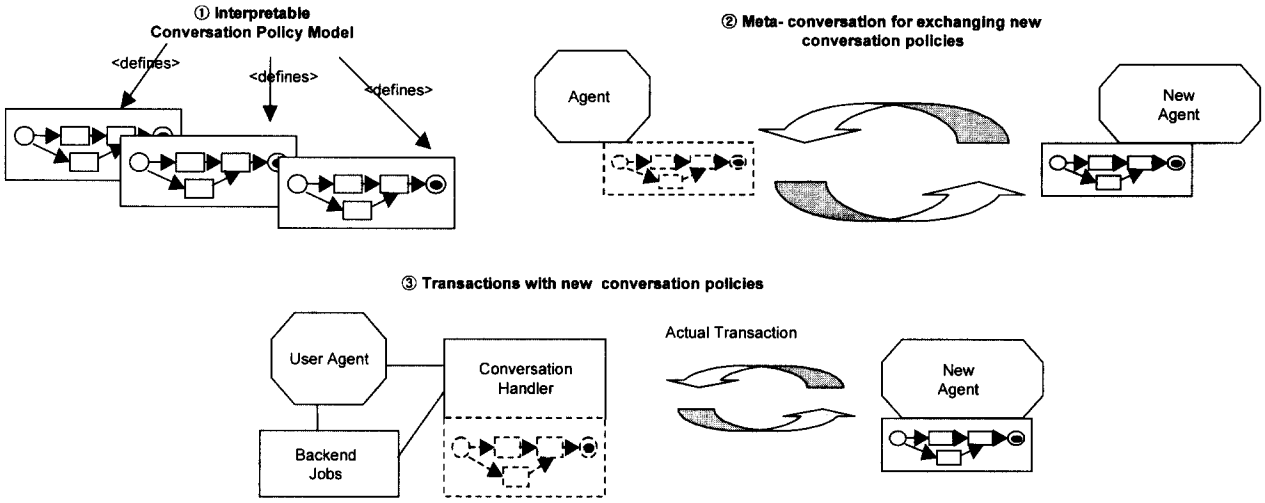
〈표 1〉 Ad Hoc 대화 정책과 관련된 연구들 요약

	논 문	요 약	한 계 점
1	Iwao et al., 2001[17]	◦ 정책 팩키지의 동적 교환 ◦ 메시지의 상태 전이 규칙의 동적 할당	◦ 작업과 정책의 분리의 한계 ◦ 임의의 종류의 메시지 처리의 문제점
2	Freire & Botelho, 2002[15]	◦ XML 기반의 대화 정책 교환 방식 ◦ 메시지 선택을 돕기 위한 규칙의 생성	◦ 작업 처리 부분과 대화 처리 부분의 인터페이스 문제 ◦ 다양한 종류의 메시지 처리에 대한 한계
3	Carabela & Beaune, 2003[10]	◦ 중앙 전략 에이전트를 통한 변화 적응 방식	◦ 중앙 에이전트의 관리와 갱신의 부담 및 어려움 ◦ 에이전트간의 협력 및 대화의 비효율성
4	Silva et al., 2003[23]	◦ 페트리넷 기반의 대화 정책 교환 ◦ 교환된 대화 정책의 오류점검 기능	◦ 교환 후 절차 미흡

3. OCAgent의 설계

3.1 개 요

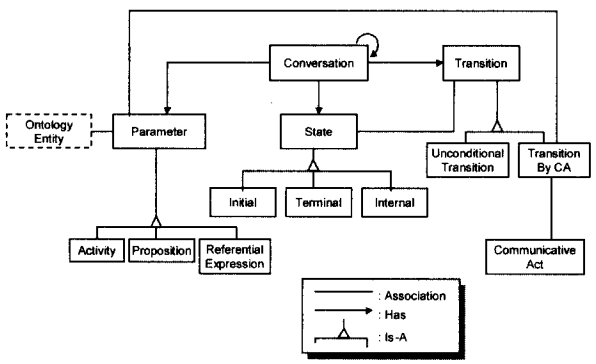
Ad Hoc한 대화 정책을 지원하기 위한 에이전트 플랫폼인 OCAgent를 설계하는 데에는 크게 세 가지 어려움이 존재한다. 첫째, 기존에 하드코딩(hard coding)되던 대화 정책을 변화가 발생했을 시에 교환 및 해석 가능하게 해야 한다. 둘째, 에이전트들간에 대화 정책을 런타임에 교환할 수 있는 미리 정해진 프로시져가 존재해야 한다. 셋째, 대화 정책의 교환이 최종 목적이 아니므로 교환된 대화 정책을 기반으로 에이전트들이 실제 자동화된 거래 행위를 할 수 있어야 한다. 이 세 가지 문제점을 해결하기 위해서 OCAgent는 (그림 3)의 세 가지 구성 요소로 이루어져 있다. 첫째 구성 요소는 해석 및 교환을 위한 대화 정책 인스턴스(instance)들을 정의하는 데 사용된다. 둘째 구성 요소는 메타 대화 절차라고 이름 붙여졌으며 런타임에 대화 정책을 교환하는 데 사용된다. 세 번째 구성 요소는 에이전트 시스템의 대화 처리 부분과 실제 작업 처리 function을 분리하는 데 사용된다. 다음 절들에서는 각각의 요소들을 하나씩 설명한다.



(그림 3) OCAgent의 개요

3.2 교환 및 해석을 위한 대화 정책 모형

기존에 존재하는 대화 정책 모형들은 대체로 대화의 상태를 표시하는 스테이트(state)와 메시지 송수신에 의해 변하게 되는 상태의 전이인 트랜지션(transition)으로 구성된다[16]. (그림 4)의 OCAgent의 대화 정책 모형도 이러한 큰 틀을 따르게 된다. OCAgent의 대화 정책 모형이 기존의 대화 정책 모형들과 다른 점은 트랜지션을 일으키는 메시지들에 대한 정보를 보다 구체적으로 명세하게 한 것이다. (그림 4)의 좌측에 나온 것처럼 메시지들의 각 파라미터는 응용 도메인의 온톨로지의 엔티티와 결합되어 구체적으로 명세된다. 또한 메시지의 파라미터가 메시지의 의미(semantics)에 따라서 작업 타입(activity), 명제 타입(proposition), 그리고 참조 표현 타입(referential expression) 등 세 가지로 분류되어 명세된다. 이는 거래방식의 변화가 발생할 경우 어떤 종류의 메시지들을 다루게 될 지 구현 시점에 알 수 없기 때문에 실행 시점에 메시지들의 해석을 용이하게 하기 위함이다. 예를 들어, 실행 시점에 주문 처리 결과에 대해 Inform 형식의 메시지를 보내야 한다면 이를 다루는 에이전트는 '주문 처리 결과'라는 '명제 타입'의 온톨로지 개체에 해당하는 정보를 상대방에게 보내야 하며, 이러한 요구사항이 OCAgent의 대화 정책 모형에 반영되어 있다. 이는 기존의 하드코딩 방식과 다른데 기존 방식에서는 구현 시점에 쌍방의 에이전트가 미리 메시지 종류나 파라미터를 모두 알고 있으며 대화 정책의 상호 교환을 가정하지 않으므로 대화 정책 모형에서는 이를 모두 구체적으로 명세할 필요가 없다. 또한 FIPA의 추천 대화 정책들 또한 모두 범용으로 개발되어 실제 응용 환경에서는 개별 개발자가 이를 확장하고 구체화하여 사용하도록 되어 있다[13].

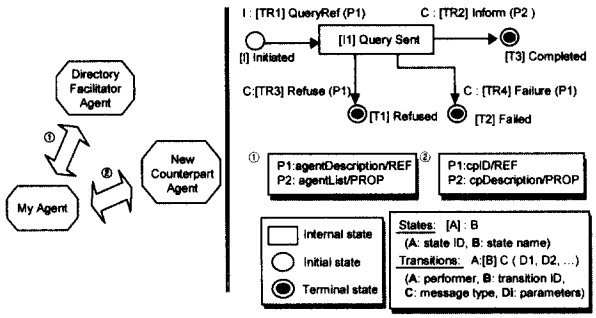


(그림 4) 교환 및 해석이 가능한 대화 정책 모형

3.3 대화 정책 교환 절차

(그림 5)는 실제 런타임시에 변화된 대화 정책을 사용하는 에이전트와 협력하게 되었을 때 대화 정책을 교환하는 절차를 보여주고 있다. (그림 5)의 왼쪽 ①의 MyAgent는 디렉토리 기능을 하는 Directory Facilitator(DF) 에이전트를 통해서 원하는 상대 에이전트(Counterpart Agent)를 검색하게 된다. 이때, 상대 에이전트가 Ad Hoc한 대화 정책을 이용하여 서비스를 제공하거나 작업의 수행을 요구한다면 My Agent는 상대 에이전트가 사용하는 Ad Hoc 대화 정책을 메타 대화를 통해 받아들이게 된다. 이때에 (그림 5)의 우측에 나와 있는 대화 정책이 ①과 ② 과정 모두에 사용되는데 이 대화 정책은 FIPA의 Query 형식과 같은 대화 정책이며 각각의 두 단계에서 DF 에이전트를 조회하기 위해, 그리고 Ad Hoc한 대화 정책을 사용하는 상대방 에이전트를 조회하기 위해 각각 다른 메시지 파라미터 세트를 이용해 사용된다. 이 Query 형식의 대화 정책에서는 I 상태(Initiated)에서 첫째로 대화의 개시자(Initiator)가 단일 질의를 의미하는 QueryRef 형식의 메시지를 P1 파라미터와 함께 대화의 상대자(Counterpart)에게 보내며 이때 대화의 상태는 I1의 QuerySent 상태가 된다. 둘째로 대화의 상대자는 질의의 내용에 따라 질의에 대한 응답을 거부하며 Re-

fuse 메시지를 보내어 T1 상태에서 대화가 종료되게 하거나, 질의 처리의 실패로 인해 T2 상태에서 종료되도록 하거나, 혹은 성공적으로 질의를 처리하여 정보 전달을 의미하는 Inform 메시지를 P2 파라미터와 함께 개시자에게 보냄으로써 대화를 T3 상태에서 종료시킬 수 있다. 이러한 절차는 대화 정책을 교환하기 위해 수행되는 또 다른 대화이므로 이러한 과정을 메타 대화라고 명명하였다. 이 때 3.2절의 대화 정책 모형으로 정의된 대화 정책 인스턴스들을 에이전트 메시지의 형식으로 변환하여야 하며 FIPA의 ACL 형태로 변환된 예가 (그림 6)에 나타나 있다.



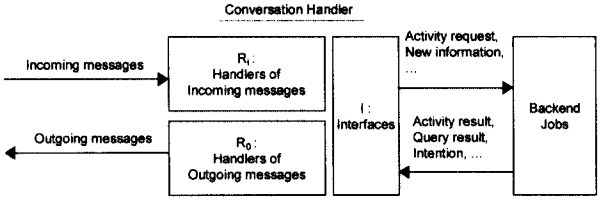
(그림 5) 대화 정책 교환을 위한 메타 대화 절차

```
(inform
  : sender PC_system_assembly_agent
  : receiver CPU_agent_02
  : content (
      (conversation_policy
      : id OrderRequestNewCPU
      : name place_order_with_credit_check
      : description "The initiator agent orders products from a supplier. Supplier agent checks the credit status of the
            buyer before confirming the order"
      : state (set (state : id i : type INI)        (state : id i1 : type INT)        (state : id i2 : type INT)
            (state : id i3 : type INT)        (state : id t2 : type TERM)        (state : id t1 : type TERM)
            (state : id t3 : type TERM)        )
      : transition (set
          (transition : id tr1 : from i : to i1 : performer INI : CA request : parameter p1)
          (transition : id tr2 : from i1 : to i2 : performer COUNTER : CA queryRef : parameter p2)
          (transition : id tr3 : from i2 : to i3 : performer INI : CA inform : parameter p3))
          ...
      )
      : parameter (set (parameter : id p1 : content order_request : ontology VSP_for_computers : type ACT)
                  (parameter : id p2 : content credit_identification_no : ontology VSP_for_computers : type REF)
                  ...
      )
    )
)
```

(그림 6) 메타 대화에서 사용되는 예제 메시지

3.4 Ad Hoc 대화 정책을 사용한 에이전트간의 협력

3.2절과 3.3절은 대화 정책을 정의하고 이를 교환하는 부분을 설명하였으나 OCAgent의 궁극적 목적은 교환 후 교환된 대화 정책을 사용하여 에이전트들이 실제 상거래 등 협력을 수행하도록 하는 것이다. 따라서 여기서는 앞서 2.1절에서 설명된 고정된 대화 정책의 문제점을 해결하기 위하여 OCAgent의 후반부의 작업 처리 부분(Backend Jobs)을 대화 정책을 처리 하는 부분(Conversation Handler)과 분리하는 점을 주로 설명한다. (그림 7)에 보이듯이 OCAgent는 작업 처리 부분과 대화 처리 부분 사이에 인터페이스를 두어서 두 부분의 상호 작용이 미리 고정된 인터페이스를 통해서만 일어나도록 한다. 상대방으로부터 메시지를 받게 되면 메시지에 해당하는 규칙이 트리거되어 메시지의 처리 결과로 해당 인터페이스의 부분이 갱신된다. 작업 처리 부분은 갱신된 인터페이스에 의해 실제 DB를 갱신하거나 기타 작업 처리 루틴을 호출하는 등의 일을 수행하게 된다. 작업 처리의 결과는 다시 반대로 인터페이스를 갱신하며, 이러한 인터페이스의 갱신에 의해 상대방에게 보내어지는 메시지가 선택되어진다.



(그림 7) 대화 처리 부분과 작업 처리 부분의 분리

OCAgent에서 사용되는 인터페이스는 기본적으로 메시지의 교환이 '작업 요청', '정보 전달', '정보 요구' 등 세 가지 유형으로 이루어진다는 점을 가정하였으며 에이전트의 내부 로직을 모델링 하기 위해 널리 쓰이는 BDI(Belief-Desire Intention) 모형을 참조하였다[2, 18]. 이때 에이전트들의

Belief는 에이전트들이 소유하는 지식의 전체 집합으로 '정보 전달', '정보 요구' 등에 의해 갱신되며, Intention은 에이전트들이 특정 목표를 성취하고자 하는 임시적인 목적으로 송신 메시지를 선택하거나 작업 요청을 수락할 때에 점검되어진다. 이 밖에 Desire는 에이전트들의 일관되고 지속되는 목표를 의미하며 하위 목표인 Intention을 생성시킨다. 에이전트들은 지속되는 Desire를 갖고 각 작업 처리마다 Intention을 생성시킴으로써 다수의 작업과 관련 대화를 동시에 수행할 수 있게 된다. 이때 모든 작업들 및 그 관련 대화들은 Belief를 공유하게 되며 각 작업 및 대화에 고유한 Belief에는 작업 인스턴스와 대화 인스턴스의 ID를 부여하여 병행 작업이 가능하도록 한다. 설계된 인터페이스들은 아래와 같다. 이때 '?'로 시작되는 부분은 인수로 쓰이는 심볼 변수 한 개를 의미하며 '$?'는 여러 개의 심볼 변수를 의미한다.

(parameter ?id ?type ?ontologyEntity $?values) : 메시지의 파라미터를 인터페이스로 전달
(proposition ?ontologyEntity $?values) : 정보 전달
(activity ?activityID $?arguments) : 작업 요청
(getValueOf ?ontologyEntity $?values) : 정보 요구
(getValueOf boolean ?ontologyEntity $?values) : Boolean 형식의 정보 요구
(actionIntention ?activityID $?arguments) : 작업 수행 의지
(informationProtectionIntention ?ontologyEntity $?values) : 정보 공유 의지

위와 같은 방식에 의해 대화 처리 부분과 작업 처리 부분이 분리되어서 1개의 고정된 작업 처리 부분이 재 구현 없이 여러 개의 변화하는 대화 정책과 함께 사용되어질 수 있다. 이제 또 하나 해결되어야 할 문제는 수신 메시지의 내용에 기반하여 어떻게 인터페이스를 갱신하고 반대로 인터페이스에 기반해서 어떻게 송신 메시지를 선택 및 구성할 것인가 하는 문제이다. 이를 위해 수신 메시지를 위한 규칙들의 집합인 Ri와 송신 메시지를 위한 규칙들의 집합인 Ro를 정의하였다. 이러한 규칙들은 FIPA의 메시지 세만틱(Semantics)에 기반하여 만들어졌다[14]. <표 2>와 <표 3>은 수신 메시지와 송신 메시지의 예를 보여주고 있다. Request 메시지를 수신하는 경우, 상대방은 수신 에이전트에게 특정 작업을 요청하게 되므로 이 부분은(*activity* ?activity $?args) 인터페이스를 생성 킨다. 이 인터페이스는 작업 처리 부분의 해당 작업을 호출하여 작업이 처리되게 된다. 마찬가지로 QueryRef 메시지는 getValueOf 인터페이스를, Inform 메시지는 proposition 인터페이스를 생성시킨다. 송신 부분에서 Request 메시지의 예를 보면, 상대방 에이전트에게 Request 메시지를 보내 작업을 요청할 때에는 해당 작업을 수행하고자 하는 의지(actionIntention)가 존재하여야 하며 또 Request 메시지가 대화의 특정 상태에서 사용 가능한 옵션 중에 하나여야 한다(messageCandidate). 이러한 조건을 만족하는 경우(messageSendable ?id)라는 결과가 생성되며, 이에 따라 상대방에게 보내는 메시지가 구축되게 된다. 따라서 이와 같은 규칙들을 지원하고자 하는 모든 종류의 메시지에 대해 정의해 줌으로써 런타임에 맞닥뜨리는 변화에 있어서 임의의 메시지 처리를 새로운 구현 없이 할 수 있게 된다.

〈표 2〉 수신 메시지 처리를 위한 규칙의 예(Ri)

메시지 종류	규칙 조건부	규칙 실행부
r_{i1} Request	(message (msgType request) (parameterID ?p)) ∧ (*parameter* ?p ACT ?activity $?args)	(*activity* ?activity $?args)
r_{i2} QueryRef	(message (msgType queryRef) (parameterID ?p)) ∧ (*parameter* ?p REF ?ref $?args) ∧ (~ (*proposition* ?ref $?args $?args2))	(*getValueOf* ?ref $?args)
r_{i3} Inform	(message (msgType inform) (parameterID ?p)) ∧ (*parameter* ?p PROP ?ref $?args)	(*proposition* ?ref ?queryRefID)

〈표 3〉 송신 메시지 구성을 위한 규칙의 예(Ro)

메시지 종류	규칙 조건부	규칙 실행부
r_{o1} Request	(messageCandidate (msgID ?id) (msgType request) (parameterID ?p)) ∧ (*parameter* ?p ?pType ?ontologyEntity $?vals) ∧ (*actionIntention* ?ontologyEntity $?vals)	(messageSendable ?id)
r_{o2} QueryRef	(messageCandidate (msgID ?id) (msgType queryRef) (parameterID ?p))	(messageSendable ?id)
r_{o3} Inform	(messageCandidate (msgID ?id) (msgType inform) (parameterID ?p)) ∧ (*parameter* ?p ?pType ?ontologyEntity $?vals) ∧ (*proposition* ?ontologyEntity $?vals $?vals2) ∧ (~ (*informationProtectionIntention* ?ontologyEntity $?vals)) ∧ (~(*informationProtectionIntention* ?ontologyEntity))	(messageSendable ?id)

다음은 인터넷을 통한 여객 티켓 예매의 예를 통해 각각 메시지가 들어올 경우 (A), 나가는 메시지의 선택 부분 (B), 그리고 작업 처리 부분 (C)에 대한 간략한 처리 절차를 설명한다.

A. 메시지가 들어올 경우,

1. OCAgentKernel에서 받은 메시지가 해석(parse)되며 결과가 팩트의 형태로 바뀐다(Part 1). 이때 Part 2는 에이전트가 기존에 갖고 있던 Belief를 나타내며 현재

예약 가능한 여행 스케쥴들을 나타낸다.

2. 1의 결과에 의해 R_i 중 받은 메시지에 해당하는 규칙 r_{ij}가 트리거된다(Part 3).
3. r_{ij}의 규칙 실행 부분(right hand side)이 실행되어 결과적으로 인터페이스가 생성된다(e.g. activity, proposition, getValueOf… 등).

```
;; Part 1 - Facts for an incoming message and parameter
(message (msgType queryRef) (parameterID p2) )
(parameter p1 REF travelItem 2003/03/19/1400 - SEOUL - TAEJEON - bus - UDG)
;; Part 2 - Bus agent's belief : Available travel schedules
(proposition travelItem 2003/03/19/1330 - SEOUL - TAEJEON - bus - IBN 45 31)
(proposition travelItem 2003/03/19/1400 - SEOUL - TAEJEON - bus - UDG 27 15)
(proposition travelItem …)
;; Part 3 - Message Handler rule - r12
(message (msgType queryRef) (parameterID p1)) AND
(parameter p1 REF travelItem 2003/03/19/1400 - SEOUL - TAEJEON - bus - UDG
    $?args) AND
(NOT (proposition travelItem 2003/03/19/1400 - SEOUL - TAEJEON - bus - UDG
    $?args))
⇒
(getValueOf travelItem 2003/03/19/1400 - SEOUL - TAEJEON - bus - UDG))
```

B. 나가는 메시지의 선택 부분

1. OCAgentKernel이 현재 대화의 상태에서 상대방에게 보내질 수 있는 메시지들의 집합을 생성한다(Part1의 messageCandidate들).
2. 각 messageCandidate는 R_o중 해당 규칙 r_{ik}를 트리거한다(Part 2).
3. r_{ik}의 조건 부분(left hand side)이 충족될 경우 해당 메시지는 messageSendable의 형태로 변환되고 최종 선택되어 보내지게 된다(Part 3). 이때 두개 이상의 메시지가 messageSendable이 될 경우는 예외 상황 혹은 에러로 처리된다.

```
;; Part 1 - Candidates for outgoing message
(messageCandidate (msgID m1) (msgType inform)(parameterID p6))
(messageCandidate …)
(messageCandidate …)
(parameter p6 PROP ticketInfo)
;; Part 2 - Verifying a message candidate - r63
(messageCandidate (msgID m1) (msgType inform) (parameterID p6)) AND
(parameter p6 PROP ticketInfo) AND
(NOT (informationProtectionIntention ticketInfo)) AND
(proposition ticket 2003/03/19/1400 - SEOUL - TAEJEON - bus - UDG ADULT 2)
=>
(messageSendable m1)
;; Part 3 - Creating an outgoing message - r10
(messageSendbale m1) AND
(NOT (AND (messageSendable m1) (messageSendable ?msg2))) AND
(messageCandidate (msgID m1) (msgType inform) (parameterID p6))
=>
(outgoingMessage (msgType inform) (parameterID p6))
```

C. 작업 처리 부분

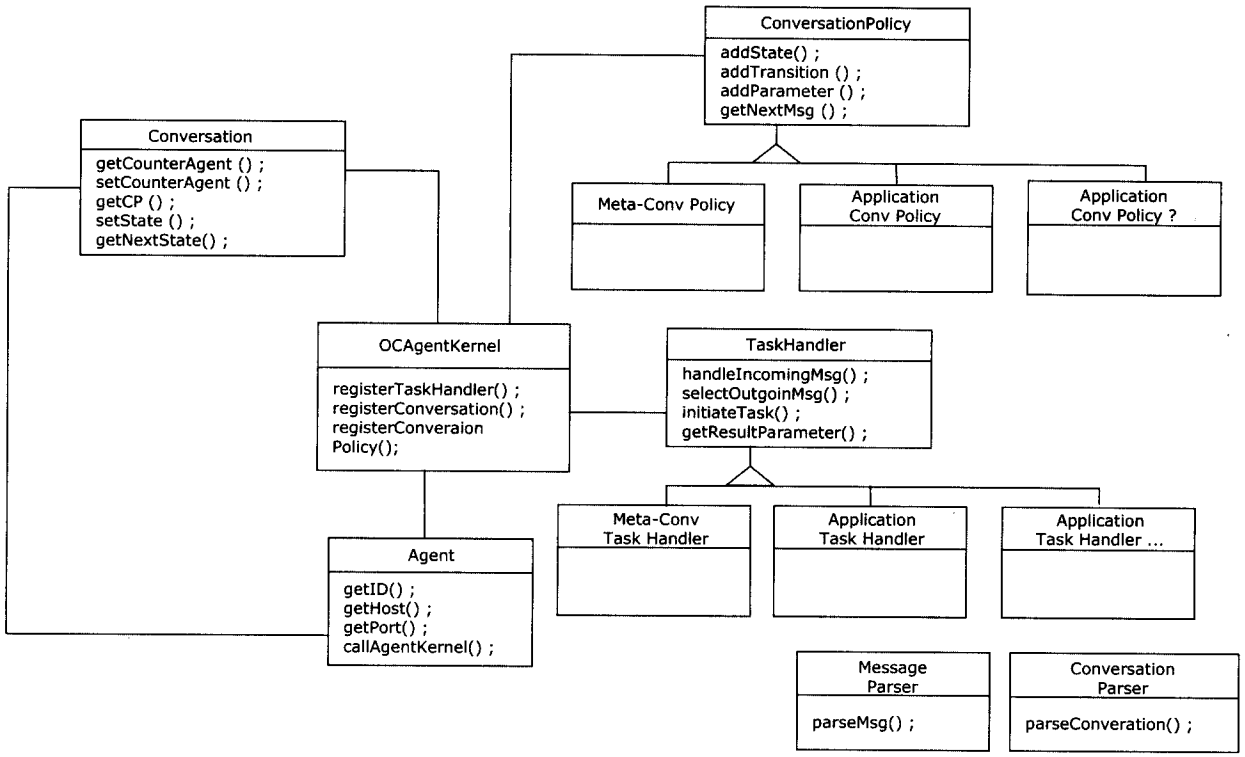
1. 새로운 메시지를 받게 되었을 때 앞서 제시된 알고리즘에 의해 activity, getValueOf 등 인터페이스가 생성되게 된다.
2. 이러한 인터페이스는 기존에 에이전트가 갖고 있는 Belief, Intention과 더불어 Part 1에 나타난 것처럼 작업처리 루틴의 조건부(left hand side)를 만족시킬 경우 실제 물리적 작업을 트리거한다. 작업 결과로서 예에 나타난 proposition과 같은 새로운 Belief를 생성시킬 수 있다.

```
;; Part 1 - Task handler for reservation request
(activity reservationRequest 2003/03/19/1400 - SEOUL - TAEJEON - bus - UDG
    ADULT 2) AND
(activityIntention reservationRequest) AND
(proposition travelItem 2003/03/19/1400 - SEOUL - TAEJEON - bus - UDG 27
    ?seatAvail) AND (> (?seatAvail 2) )
=>
(… physical_activity_for_reservation …)
(proposition ticketInfo ticket 2003/03/19/1400 - SEOUL - TAEJEON - bus - UDG
    ADULT 2)
```

4. 프로토타입 시스템

4.1 OCAgent의 구조

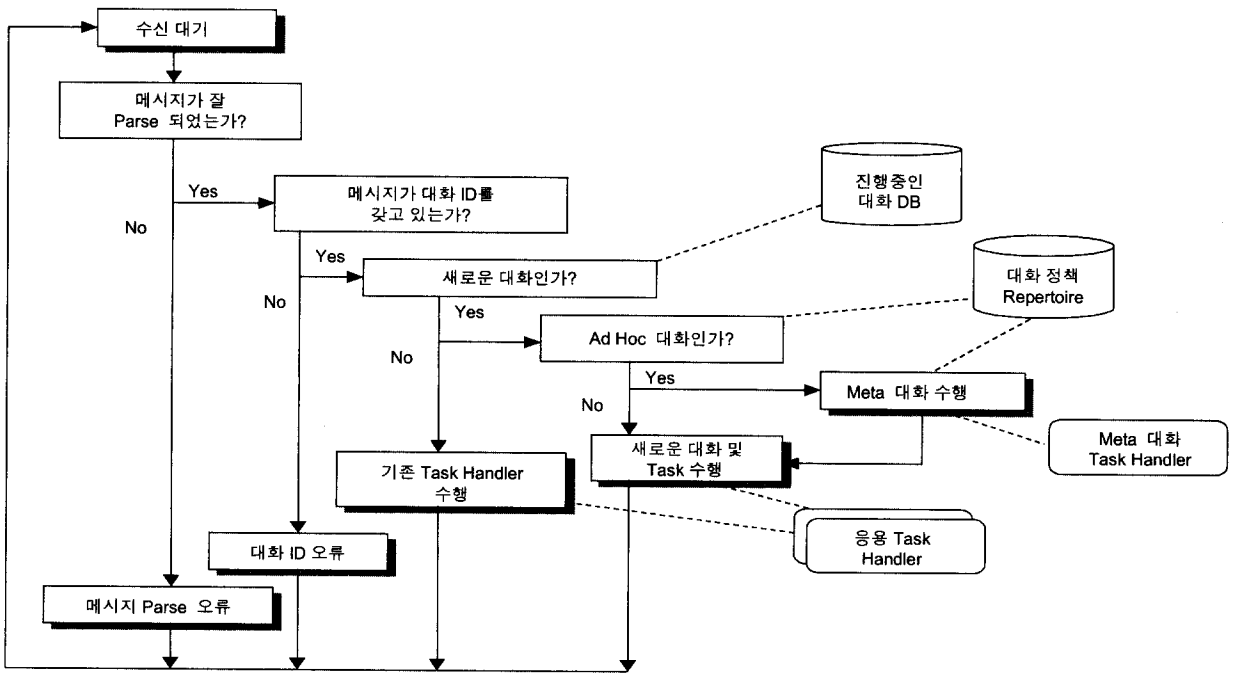
3장에서 설명된 OCAgent의 주요 구성요소의 설계에 따라 OCAgent의 프로토타입이 구현되었다. 프로토타입 시스템은 크게 대화를 처리하는 부분, 작업을 처리하는 부분, 그리고 메시지 및 대화 정책 인스턴스를 해석하는 부분(parser)들로 구성되어졌다. OCAgent의 주요 객체들이 (그림 8)에 나타나 있다. OCAgent의 핵심 부분은 OCAgentKernel에 구현되어 있으며 OCAgentKernel에 개별 에이전트가 수행하는 작업에 관해 TaskHandler 클래스를 상속하는 객체들을 추가해 주면 응용 영역을 위한 에이전트 시스템이 개발된다. OCAgent는 그러나 기본적으로 메타 대화를 처리하기 위한 TaskHandler 및 대화 정책은 포함하고 있다. 메타 대화를 통해 습득하는 대화 정책 인스턴스나 응용 작업을 위한 대화 정책 인스턴스는 ConversationPolicy 클래스를 상속하여 정의된다. 실제 대화가 시작되면 각 개별 대화의 상태를 추적하고 관련 정보를 저장하기 위한 Conversation 객체가 생성된다. 이러한 객체들은 대화 상대의 정보를 포함하는 Agent 클래스의 객체들과 연관성을 갖는다. 전체적인 프로토타입 시스템은 JAVA로 구현되었으며 대화처리 부분의 규칙 및 인터페이스 부분은 규칙기반 추론 엔진인 JESS를 활용하여 구현되었다[11]. JESS로 정의된 규칙의 예가 부록에 제시되어 있다.



(그림 8) OCAgent의 객체 모형

4.2 OCAgentKernel의 논리 구조

OCAgent의 핵심 부분인 OCAgentKernel 부분은 (그림 9)에 도식화된 논리 흐름에 따라 수행된다. 전체적으로 OCAgent-Kernel은 수신 대기와 수신 메시지에 따른 메시지 처리를 수행하는 루프를 반복 수행하게 된다. 또한 특정 에이전트가 여러 대화를 동시에 수행할 수도 있으므로 진행중인 대화들을 관리하게 되며(진행중인 대화 DB), 자신이 알고 있는 대화 정책의 리스트를 관리하게 된다(대화 정책 Reper-

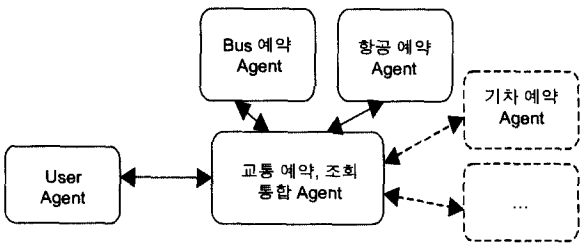


(그림 9) OCAgent의 논리 구조

toire). 메시지가 도착하면 이를 해석하며 메시지에 오류가 없는지, 적절한 대화 ID를 갖고 있는지 점검한다. 메시지가 적절할 경우 이 메시지가 새로운 대화를 위한 메시지인지 기존에 진행되던 대화의 후속 메시지인지를 점검하여, 기존 대화의 메시지라면 실행중이던 응용 TaskHandler를 호출하게 된다. 만약 새로운 대화라면 대화와 관련된 대화 정책이 Ad Hoc 타입인지 그렇지 않은지 대화 정책 Repertoire를 검사하게 되며, 만약 Ad Hoc한 타입의 새로운 대화 정책이라면 메타 대화 TaskHandler를 통해서 해당 대화 정책을 습득한 후에 관련된 응용 TaskHandler를 호출하게 된다.

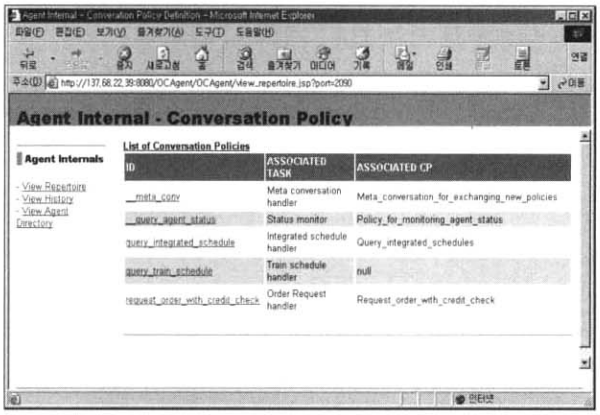
4.3 예제 화면

다음으로 에이전트 플랫폼 내부에서 일어나는 일들을 모니터링 하기 위한 간단한 화면들을 다음과 같이 제시한다. 예를 들어 (그림 10)과 같이 버스, 기차, 항공 등 운송수단에 대한 예약 서비스를 하는 멀티 에이전트 시스템이 있다고 가정하자. 이 때에 이 서비스에는 새로운 운송 수단의 추가, 새로운 노선의 추가, 혹은 기존 거래 방식의 변경 등에 의해 Ad Hoc한 대화 정책이 등장할 수 있다. 이 경우에 서비스를 사용하는 에이전트가 이러한 새로운 대화 정책에 대해 적응할 수 없다면 예약을 위한 프로그래밍 루틴은 변화된 대화 정책을 이용해 거래 및 조회를 할 수 없게 된다.

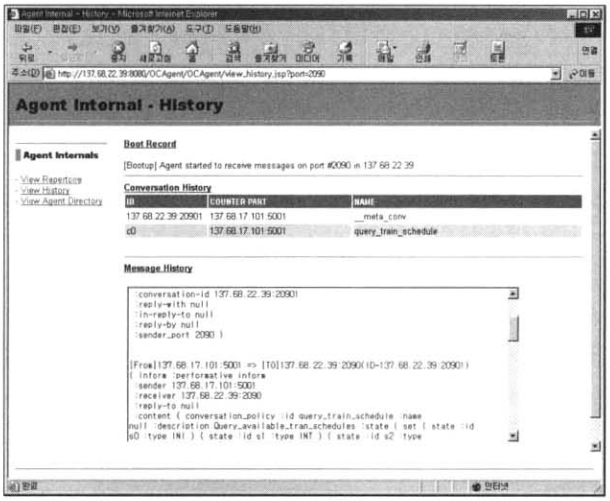


(그림 10) 교통 예약, 조회 에이전트와 환경 변화

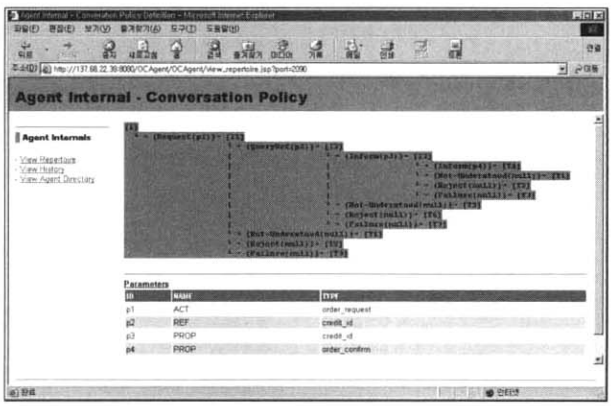
(그림 11)은 특정한 에이전트가 처리 가능한 대화 정책 인스턴스의 리스트를 보여주는 화면이다. 여기에는 교통 예약 조회 통합 에이전트의 다섯 가지 대화 정책이 표시되어 있으며 상위 2개의 대화 정책은 시스템에 내재된 대화 정책으로 메타 대화와 에이전트 모니터링을 위해 사용되는 정책이다. 아래 셋은 각각 User 에이전트의 통합 조회에 대응하기 위한 대화 정책, 기차 예약을 위해 기차 예약 에이전트와 대화하기 위한 대화 정책, 그리고 사용자의 예약 주문을 처리하기 위한 대화 정책이다. 운송 수단의 예약을 위해 사용되는 응용 대화 정책으로 이때 query_train_schedule에는 해당하는 대화 정책이 빈 값(null)으로 돼 있음을 확인할 수 있으며 이 경우엔 메타 대화를 하기 이전으로 해당 대화 정책을 기차 예약 에이전트로부터 아직 습득하지 못한 경우이다. 이러한 상태에서 기차 예약 에이전트와 대화를 시도하게 되면 OCAgentKernel에 등록된 metaTaskHandler가 실행되어 메타 대화가 우선 수행되어 query_train_schedule을 위한 Ad Hoc 대화 정책을 습득하게 된다. 이러한 과정이 에이전트의 내부에서 일어나는 대화 및 메시지 교환의 히스토리를 나타내 주는 화면인 (그림 12)에 나타나 있다. 대화 히스토리를 보면 우선 메타 대화가 수행



(그림 11) 사용 가능한 대화 정책의 리스트를 보여주는 화면



(그림 12) 대화 및 메시지 교환의 히스토리를 보여주는 화면



(그림 13) 특정 대화 정책 인스턴스의 구조를 트리형태로 보여주는 화면

되고 그 이후에 실제 기차 스케쥴을 조회하기 위한 대화가 수행되었음을 알 수 있다. (그림 13)은 에이전트가 처리할 수 있는 대화 정책의 리스트 중에서 특정한 대화 정책의 구조를 트리 형태로 보여주는 화면이다. 이 예에서는 실제 운송수단에 신용카드를 이용하여 예약을 하기 위한 대화 정책을 나타내고 있으며 이를 위해 예약 요청, 신용 정보 등에 관한 메시지의 교환을 정의하고 있다.

4.4 기존 연구와의 비교

<표 4>는 몇 가지 대표적인 에이전트 플랫폼들의 특성을 보여주고 있다. <표 4>에 나온 바와 같이 기존의 에이전트 플랫폼은 FIPA의 표준 대화 정책이나 사용자 정의 대화 정책은 지원하지만 Ad Hoc한 대화 정책은 지원하지 않고 있다. 그러나 본 논문에서 제시된 OCAgent의 메커니즘을 기존의 에이전트 플랫폼을 확장하여 유사하게 구현하는 것은 가능할 수 있다. 예를 들어 JADE의 FSMBehaviour를 활용하여 OCAgent의 메타 대화 절차를 구현하고 교환된 대화 정책을 저장하는 저장소 기능 및 메시지 처리 규칙 (R_o와 R_i) 등을 적용하면 OCAgent와 유사한 기능을 제공할 수 있을 것으로 기대된다.

〈표 4〉 대표적 에이전트 플랫폼들의 대화 지원 특성

	대화 정책 지원		
	내재된 대화 정책	사용자 정의 대화 정책	Ad Hoc 대화 정책의 지원 가능성
JADE [6]	FIPA 표준 대화 정책	지원. 일반 대화 정책 Class를 상속하여 만들 수 있음. (예) AchieveREInitiator, AchieveREResponder modules	지원하지 않음. (FSMBehaviour 클래스를 수정하여 OCAgent 기능을 구현해야 함)
FIPA OS [22]	FIPA 표준 대화 정책	지원. Conversation module을 수정하여 지원함.	지원하지 않음. (ConversationManager module의 수정 필요.)
Zeus [20]	Contract Net, 경매 등	지원. 외부 프로그램으로 작성 후 Zeus 모듈로 등록함.	지원하지 않음.

다음으로, 앞서 <표 1>에서 소개된 네 가지 연구와의 차별점은 다음과 같이 정리될 수 있다. 첫째, 네 가지 연구 모두에서 에이전트의 대화 처리 부분과 작업 처리 부분이 어떻게 분리되어질 수 있는지 명확히 제시되지 않고 있으나 본 연구에서는 인터페이스를 사용하여 두 부분의 분리 방식을 제시하였다. 둘째, 기존 연구들에서 미흡한 부분인 초기 구현 단계에서 미리 알 수 없는 종류의 메시지에 대한 대응을 위해 각 메시지에 해당하는 규칙을 만들고 이러한 규칙들이 인터페이스를 사용하여 작업 처리 부분과 연계되도록 하였다. 셋째, 세 번째 연구에서처럼 중앙 집중식 변화 대응 방식을 사용하지 않고 각 개별 에이전트들이 1대 1의 상호작용을 통해 변화에 대응할 수 있도록 하였다. 넷째, 네 번째 연구의 경우에 대화 정책 교환 후에 실제 상거래를 할 수 있는 절차를 제시하고 있지 않으나 본 연구에서는 메시지 처리 규칙과 인터페이스 등을 이용하여 실제 교환 후, 즉 메타 대화 후의 절차가 어떻게 이루어지는지를 명확히 나타내고 있다(3.4절).

5. 토론 및 결론

본 연구에서는 동적인 멀티 에이전트 시스템 환경에서 거래 방식 등의 변화에 의해 수반되는 Ad Hoc한 대화 정책에 적응성을 갖는 에이전트 플랫폼이 설계 되었으며 이를 기반으로 프로토타입 시스템이 구축되었다. 본 연구에서 제시된 방식의 적응성은 기존의 재래적 시장 방식보다 훨씬 빈번한 변화를 수반하는 전자 시장 환경의 크고 작은 변화에 에이전트 플랫폼을 다시 구현해야 하는 부담을 덜어주어 결과적으로 기업들의 정보 시스템 유지 관리 비용을 낮추고 기업들이 시장 변화에 더욱 신속히 대응할 수 있게 해 준다.

본 연구에서 제시된 OCAgent는 적응성 측면에서 큰 장점을 갖고 있으나 몇 가지 한계점 또한 가지고 있다. 첫째, OCAgent가 모든 종류의 변화에 대해 적응성을 갖는 것은 아니다. 응용 도메인의 온톨로지에 변화가 생기거나 혹은 도메인에서 처리해야 할 작업이 새로이 바뀌는 경우, 혹은 작업의 형태가 크게 변화하는 경우에는 적응이 가능하지 않다. 둘째, 에이전트 플랫폼의 구성이 복잡해지며 따라서 에이전트 플랫폼의 구현도 복잡해진다. 그러나 적응성이 더욱 중요한 동적인 환경에서는 그러한 복잡성에 의한 비용보다 적응성이 주는 이득이 더욱 크게 되므로 제시된 접근 방법이 유용할 것으로 기대된다.

추후 연구과제로는 좀 더 확장성 있고 응용 에이전트의 개발이 용이한 형태의 개발 환경 구축과 다양한 응용 환경에 OCAgent를 적용하는 것을 고려하고 있다.

참 고 문 헌

[1] 김종완, 김영순, "전자상거래에서 제품 정보 추천을 위한 멀티 에이전트 시스템의 워크플로우 구축", 정보처리학회논문지B, 제8-B권 제6호, pp.617-624, 2001.

[2] 이명진, 김진상, "BDI 에이전트 환경에서 협상을 위한 에이전트 통신 언어", 정보처리학회논문지B, 제10-B권 제1호, pp. 21-26, 2003.

[3] 황현아, 임한규, "교통편 예약 에이전트 시스템 설계 및 구현", 정보처리학회논문지D, 제10-D권 제1호, pp.125-132, 2003.

[4] Ahn, H. J., Lee, H., Park, S. J., "A Flexible Agent System for Change Adaptation in Supply Chains," Expert Systems with Applications, Vol.25, No.4, 2003.

[5] Ahn, H. J., Lee, H., Yim H. and Park, S. J., "Handshaking Mechanism for Conversation Policy Agreements in Dynamic Agent Environment," *Proceedings of the AAMAS Workshop on Agentcities : Challenges in Open Agent Systems*, pp.46-50, 2002.

[6] Bellifemine, F., Poggi, A. and Rimassa, G., "Developing multi agent systems with a FIPA-compliant agent framework," *Software -Practice And Experience*, Vol.31, pp.103-128, 2001.

[7] Bertolotto, M., O'Hare, G., Strahan, R., Brophy, A., Martin, A. and McLoughlin, E., "Bus Catcher : a Context Sensitive Prototype System for Public Transportation Users," *Proceedings of the Third International Conference on Web Information Systems Engineering (Workshops)-(WISEw '02)*, pp.64-72, 2002.

[8] Brenner, W., Zarnekow, R. and Hartmut, W., *Intelligent Software Agents*. Berlin : Springer Verlag, 1998.

[9] Camacho, D., Molina, J. M. and Borrajo, D., "A multiagent approach for electronic travel planning," *In Proceedings of the Second International Bi-Conference Workshop on Agent Oriented Information Systems (AOIS 2000), Austin, TX (USA), AAAI-2000.*

[10] Carabelea C, Beaune, P., "Engineering a Protocol Server Using Strategy - Agents," *Proceedings of CEEMAS2003* 413-422, 2003.

[11] Carlson, D., "Rules and Web - Object Systems," *Object Magazine*, June, 1998.

[12] Elio R. and Haddadi, A., "On abstract task models and conversation policies," *Working Notes of the Workshop on Specifying and Implementing Conversation Policies, Seattle, Washington*, pp.89-98, 1999.

[13] FIPA00025, "FIPA Interaction Protocol Library Specification," *Foundation for Intelligent Physical Agents*, 2001.

[14] FIPA00037, "FIPA Communicative Act Library Specification," *Foundation for Intelligent Physical Agents*, 2002.

[15] Freire, J. and Botelho, L., "Executing Explicitly Represented Protocols," *Proceedings of the AAMAS Workshop on Agentcities : Challenges in Open Agent Systems*, pp. 37-40, 2002.

[16] Greaves, M., Holmback, H. and Bradshaw, J., "What is a conversation policy?," *Working Notes of the Workshop on Specifying and Implementing Conversation Policies, Seattle, Washington*, pp.1-10, 1999.

[17] Iwao T, Wada Y, Okada M, Amamiya M., "A Framework for the Exchange and Installation of Protocols in a Multi-Agent System," *Proceedings of CIA 2001*, pp.211-222, 2001.

[18] Jennings, N. R., Sycara, K. and Wooldridge, M., "A roadmap of agent research and development," *Autonomous Agents and Multi Agent Systems*, Vol.1, No.1, pp.275-306, 1998.

[19] Labrou, Y., Finin, T. and Peng, Y., "Agent communication languages : the current landscape," *IEEE Intelligent Systems*, Vol.14, No.2, pp.45-52, 1999.

[20] Nwana, H. S., Ndumu, D. T. and Lee, L. C., "ZEUS : An Advanced Tool Kit for Engineering Distributed Multi-Agent Systems," *Proceedings of the Practical Application of Intelligent Agents and Multi -Agent Systems, London, UK*, pp.377-391, 1998.

[21] Pitt. J. and Mamdani, A., "Communication protocols in multi-agent systems," *Proceedings of the Workshop on Specifying and Implementing Conversation Policies*, pp. 39-48, 1999.

[22] Poslad, S., Buckle, P. and Hadingham, R., "The FIPA - OS agent platform : Open Source for Open Standard. *Proceedings of the PAAM2000, Manchester, UK*, pp.355-368, 2000.

[23] Silva, L. P., Winikoff, M. and Liu, W., "Extending Agents by Transmitting Protocols in Open Systems," in the proceedings of the Challenges in Open Agent Systems '03 workshop held in Melbourne, July, 2003.

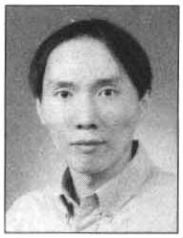
[24] Singh, M. P., "Agent communications languages : rethinking the principles," *IEEE Computer*, Vol.31, No.12, pp.40-47, 1998.

부록. JESS 규칙의 예

```
;; 수신된 inform 형식의 메시지 처리
(defrule informMsgHandler
 (message (msgType inform) (parameterID ?p) )
 (parameter ?p ?ref $?args)
⇒
  (assert (addBelief ?ref ?args))
)

;; Inform 형식의 메시지를 발신할 것인지 체크
(defrule informMsgChecker
 (messageCandidate (msgID ?id) (msgType inform)
 (parameterID ?pid) )
```

```
  (parameter ?pid ?pType ?ontologyEntity $?vals)
  (not (informationProtectionIntention ?ontologyEntity
  $?vals))
  (not (informationProtectionIntention ?ontologyEntity ) )
  (proposition ?ontologyEntity $?vals $?vals2)
=>
  (assert (messageSendable ?id) )
)
```



안 형 준

e-mail : hjahn@isir.kaist.ac.kr
1995년 KAIST 경영과학 학사
1997년 KAIST 경영공학 석사
2004년 KAIST 경영공학 박사과정
현재 경영정보시스템, 전자상거래 전공
관심분야 : Multi-Agent System, 전자상거래, 지능형 정보기술