다중 에이전트 시스템 구축을 위한 아키텍처 개발방법 및 지능형 교통 시스템에의 응용

(An Architecture Method for Multi-Agent System Developments and its Application to Intelligent Transport Systems)

이 승 연 † 박 수 용 †† 정 성 원 †††

(Seungyun Lee) (Sooyong Park) (Sungwon Jung)

요 약 본 논문은 다양한 종류의 분산 인공지능 문제들을 에이전트라는 추상적 단위와 에이전트간의 상호작용을 토대로 해결하는 다중 에이전트 시스템을 개발하는 체계적 접근방법으로서 개발 방법론의 핵심인 아키텍처의 개발방법을 제안한다. 목표를 기반으로 문제영역을 이해하고, 여기에서 추출된 에이전트들을 이용하여 시스템을 개발함에 있어 지침이 되는 아키텍처 개발공정을 다중 에이전트 시스템의 특성인 조정과 자율성을 고려하여 제안한다. 각 관점마다 적용될 수 있는 아키텍처 스타일과 패턴들을 정의하고, 제안한 아키텍처를 UML(Unified Modeling Language)을 이용하여 표현하며, 아키텍처를 설명하는 ADL(Architecture Description Language)을 이용하여 정형화시킨다. 또한, 이를 지능형 교통시스템의 출발전 교통정보 안내 서브시스템에 적용, 구현함으로써, 제안하는 아키텍처를 검증해 보고, 이를 기반으로 소프트웨어를 개발하는 기초를 마련한다.

Abstract Emerging to the era of distributed software environments, the research of Multi-Agent Systems(MAS), which tries to solve complex systems with entities called agent, is on the rise. This paper proposes the architecture method for MAS, the most important phase of systematic software developments. Goal-based approach is used for the problem domain analysis, and agent is mapped to system's refined goals. Considering system's view of coordination and autonomy, architectural styles and patterns are defined to represent the architecture. UML(Unified Modeling Language) and ADL(Architecture Description Language) are used for modeling and formalizing the architecture, and finally, the proposing architecture method is applied to the domain of Intelligent Transport Systems and validated by implementing its prototype.

1. 서 론

실세계에서 발생하는 대부분의 문제들은 중앙 집중처리 방식에 의해 해결 가능하지만, 단일 프로세서만으로 해결하기 어려운 복잡한 문제들은 분산된 환경에서의 처리가 요구된다. 거대한 시스템을 개발함에 있어, 시스템의 요구사항을 정의하고 분석과 설계, 구현의 개발 과정을 모두 새로이 적용하던 기존의 소프트웨어 개발방법은 이제, 기존의 소프트웨어들을 식별하고, 이들의 간단한 커스터마이제이션과 그들간의 상호작용을 통하여 거대한 시스템의 요구사항을 충족시키는 방법으로 변화하고 있다. 이러한 변화는 인터넷의 발달과 웹의 사용이 증가됨에 따라, 웹 상에 퍼져있는 기존의 소프트웨어들과 새로이 만들어지는 소프트웨어 사이의 상호연동을 지원하는 방향으로 전개되고 있는데, 이는 각각의 컴포넌트들이 환경에 동적으로 반응하며, 능동적으로 작업을 수행해 나갈 수 있는 기능을 요구한다.

분산된 환경에서 동적인 작업을 능동적으로 수행하는 소프트웨어를 지원하기 위하여, 분산 인공지능에서는 다중 에이전트 시스템(Multi-Agent System) 환경을 이

· 본 연구는 한국과학재단 목적기초연구(2000-1-30300-001-3)지원으로 수행되었음

† 비 회 원 : 한국전자통신연구원 컴소연 S/W공학연구부 연구원
coral@etri.re.kr
†† 정 회 원 : 서강대학교 컴퓨터학과 교수
sypark@ccs.sogang.ac.kr
††† 비 회 원 : 서강대학교 컴퓨터학과 교수
jungsung@ccs.sogang.ac.kr

논문접수 : 2001년 1월 8일
논문완료 : 2001년 5월 23일

용하여 에이전트들간의 상호 협력을 통해 문제를 해결하는 방법을 제시해 왔다. 다양한 종류의 분산 인공지능 문제들을 에이전트라는 추상적 단위의 능동적 개체와 이들간의 상호작용을 토대로 해결하려는 시도는 인터넷과 웹의 급속한 발전으로 그 연구가 활발히 진행되고 있으며, 그 중에서도 분산된 환경에 적용될 수 있는 에이전트 시스템의 모습을 제시하는 아키텍처에 대한 연구가 진행되고 있다. 그러나, 이러한 시도는 고도의 분산되고 복합적인 시스템을 분석, 설계, 구현하는 실용 소프트웨어를 효과적으로 개발하려는 소프트웨어공학의 체계적 접근방법의 부재로 아직까지 실험단계에 그치고 있는 실정이다. 아키텍쳐는 소프트웨어 시스템을 개발하는 지침을 제공하는 핵심요소이다. 모든 소프트웨어는 그에 맞는 골격을 가지고 있으며, 그 골격이 올바르게 세워져야 그것에 살을 붙이고 확장하는 것 또한 올바르게 분석, 설계될 수 있고, 이는 시스템의 신뢰도 및 성능을 높이는 지름길이 될 수 있다. 따라서 소프트웨어 공학의 입장에서 문제영역을 이해하고 이를 바탕으로 시스템의 골격을 이루는 아키텍쳐를 체계적 공정을 거쳐 개발하고, 이를 정형화하는 것이 올바른 시스템을 개발하기 위한 필수조건이라 할 수 있다. 따라서, 본 논문에서는 분산된 환경에서 작동하는 다중 에이전트 시스템을 개발하는 체계적 접근방법으로서 소프트웨어 개발 공정의 가장 핵심이 되는 아키텍쳐 개발에 대한 지침을 제안하고자 한다.

목표를 기반으로 문제영역을 이해하고, 여기에서 추출된 에이전트들을 이용하여 시스템을 개발하는데 지침이 되는 아키텍쳐를 다중 에이전트 시스템의 특성인 조정(Coordination)과 자율성(Autonomy)을 고려한 관점에서 개발하도록 하였다. 각 관점마다 아키텍쳐 개발에 필요한 요소들을 정의하고, 적용될 수 있는 아키텍쳐 스타일(Architectural Styles)과 패턴(Pattern)들을 제시하였다. 제안한 아키텍쳐를 UML(Unified Modeling Language)[1]의 표기법을 이용하여 표현하였고, 아키텍쳐를 설명하는 ADL (Architecture Description Language)[2]을 이용하여 정형화시키고 정의된 모듈들의 타입과 상호관계를 검사하였다. 또한, 이를 지능형 교통시스템(ITS)의 출발전 교통정보 안내 서브시스템(PTIS)[3] 영역에 적용하여 구현하였다.

본 논문의 구성은 다음과 같다. 2장에서는 본 연구의 배경지식과 기존의 관련연구들에 대해 살펴보고, 3장에서는 다중 에이전트 시스템을 개발하는 아키텍쳐 지침을 조정과 자율성의 관점에서 제시한다. 4장에서는 3장에서 제시된 아키텍쳐 지침을 출발전 교통정보 안내 서브시스템(PTIS)에 적용하여 그의 타당성을 검증해 보고 5장에서는 결론을 맺고자 한다.

2. 기본개념 및 기존의 관련연구

본 연구에 들어가기에 앞서, 다중 에이전트 시스템의 정의 및 성질, 아키텍쳐 개발시 고려해야 하는 요인들에 대해 살펴보고자 한다.

다중 에이전트 시스템은 에이전트 혼자만의 능력이나 지식으로는 해결하기 힘든 문제를 다른 에이전트들과 상호 협력하여 문제를 풀어나가는 시스템으로 각 에이전트의 목표와 문제 해결 상황의 차이에 따라 합의, 교섭, 설득, 경합 등의 프로세스를 통해 문제 해결을 추구하는 시스템이다[4]. 다중 에이전트 시스템은 독립적인 단일 에이전트로는 해결하기 힘든 복잡한 문제에 대한 서비스를 제공해주고, 기존 시스템과의 상호작용을 가능하게 하며, 속도, 안정성, 확장성이 용이하고 실세계의 불확실한 데이타나 지식을 처리할 수 있다는 장점이 있다. 이러한 다중 에이전트 시스템이 문제를 해결하는데 사용하는 주된 방법은 조정과 교섭, 그리고 통신이다[5]. 조정은 개개의 에이전트들이 이루는 집단이 서로 화합하여 행동하게 하는 과정이며, 교섭은 조정의 한 부분으로 둘 이상의 에이전트 사이의 분쟁을 해결하여 타협점을 찾아내는 과정을 말한다. 조정과 교섭을 위해서는 에이전트들 사이의 통신수단이 필수적이다.

Nwana와 Lee, 그리고 Jennings는 그들의 논문에서 다중 에이전트 시스템을 구성하는 에이전트들 사이의 조정을 지원하기 위해서는 상호작용을 예견할 수 있는 구조, 다양한 환경변화에도 작업을 처리할 수 있는 에이전트의 유연성, 조정 프로세스를 표현하는 시스템 전체의 구조, 그리고 이러한 구조를 이용할 수 있는 지식과 추론능력이 있어야 한다고 정의하였다[6].

다중 에이전트 시스템을 개발함에 있어, 앞에서 언급한 속성의 고려와 더불어, 시스템의 기능적, 비기능적인 측면에 대한 고려가 이뤄져야 한다. 즉, 구현된 시스템이 다중 에이전트 시스템의 속성들을 만족하더라도, 시스템의 오류가 빈번하다거나, 안정적이지 못하다면, 시스템으로서의 역할을 제대로 할 수 없기 때문이다. 그러나, 다중 에이전트 시스템을 개발하기 위하여 제안하는 기존의 아키텍쳐 연구들은 구현의 입장에서 시스템을 바라보아, 쉽게 개발할 수 있는 방향으로 다중 에이전트 시스템 아키텍쳐를 개발하였으며, 이는 시스템의 비기능적인 측면에 대한 고려가 낮았다.

SRI의 OAA(Open Agent Architecture)[7]는 프로그래밍 입장에서 보다 쉽게 구현할 수 있는 방법에 우선순

위를 두어, 중앙의 퍼실리테이터(Facilitator)에 의해 시스템을 관리하는 아키텍쳐를 제시하였는데, 중앙의 퍼실리테이터에 의한 관리는 병목현상의 문제점을 안고 있으며, 이로 인한 시스템 성능 저하 및 부분의 고장이 시스템 전체의 사용을 막는 신뢰성에 문제를 보이고 있다. RETSINA(Reusable Task Structure based Intelligent Network Agents)[8]는 웹을 기반으로 정보제공을 위하여 설계된 것으로, Matchmaker[18]라는 중재자를 여럿 두어서 에이전트 사이의 통신과 작업처리를 가능하게 하였다. 그러나, Matchmaker는 시스템 내에 존재하는 모든 에이전트에 대한 정보를 저장하고 있기 때문에 자료의 중복과 일관성을 유지하는데 많은 노력이 필요하다. 다중 에이전트 시스템의 아키텍쳐는 이러한 중복성과 일관성을 해결하는 구조로 설계되어야 한다. 앞의 두 가지 아키텍쳐와는 달리, DESIRE(a framework for DEsign and Specification of Interacting REasoning components)[9]는 객체지향의 개념을 따라 설계되었으며, 에이전트 내부 아키텍쳐에 초점을 맞추고 있다. 에이전트 내부를 여러 구현모듈로 세분화하였으며, 각 모듈에 대한 자세한 구현설명이 덧붙여 있다. 그러나, 이러한 내부 구현모듈간의 흐름이 어떻게 진행되는지 보여주고 있지 않아서, 에이전트 내부의 동적 변화에 대한 시스템 흐름을 예측하기 어려우며, 에이전트 내부에 초점을 맞추고 있어, 다중 에이전트 시스템의 주요 요소인 조정에 대한 아키텍쳐 표현이 미비하다.

기존의 아키텍쳐 연구들은 개발하고자 하는 시스템의 아키텍쳐를 설계함에 있어, 필수적으로 고려해야 하는 속성에 대한 정의가 명확히 이루어지지 못했으며, 이를 개발하는데 고려해야 하는 시스템의 비기능적 요인들에 대한 고려가 미비하였다. 다중 에이전트 시스템을 개발하기 위해서는 에이전트들 사이의 상호작용, 환경변화에 대처하는 유연성, 이를 지원하는 에이전트간 또는 에이전트 내부의 흐름을 표현하는 구조가 필요하며, 이는 시스템의 기능적인 측면뿐 만 아니라, 시스템의 성능, 신뢰성, 적응성과 같은 품질과 관련된 다양한 속성들의 고려와 함께 설계되어야 한다.

본 연구는 소프트웨어 공학의 측면에서 다중 에이전트 시스템 개발을 위한 아키텍쳐 개발방법을 제안한다. 분산된 환경에서 독립적인 단일 프로세스로는 해결하기 힘든 복잡한 문제에 대한 서비스를 제공해 주는 시스템의 기능적인 측면을 고려함과 동시에, 기존 시스템과의 상호작용을 가능하게 하며, 속도와 안정성, 확장성이 용이하게 하고 실세계의 불확실한 데이타나 지식을 처리하여 환경의 변화에 적응할 수 있는 비기능적인 속성들을 고려하여 시스템을 개발하기 위한 아키텍쳐 개발공정을 제안한다. 복잡한 시스템을 쉽게 이해할 수 있는 아키텍쳐를 제안하기 위하여 시스템을 여러 가지 관점에서 바라보고, 각 관점마다 비중을 두어 고려해야 하는 아키텍쳐 속성들을 판별하여 이를 만족시키는 아키텍쳐를 개발하기 위해 필수적인 공정을 제안한다. 아키텍쳐 개발공정의 각 단계는 여러 가지 스타일과 패턴을 사용하여 표현하고, UML과 ADL을 이용하여 정형화한다.

3. 다중 에이전트 시스템 아키텍쳐 개발 방법

본 장에서는 다중 에이전트 시스템을 위한 아키텍쳐 개발방법을 제안하고 이를 아키텍쳐 속성을 고려한 다양한 관점에서 조명해 본다.

3.1 전체공정

다중 에이전트 시스템을 개발하는데 핵심인 아키텍쳐는 그림 1과 같은 공정을 거쳐 개발된다.

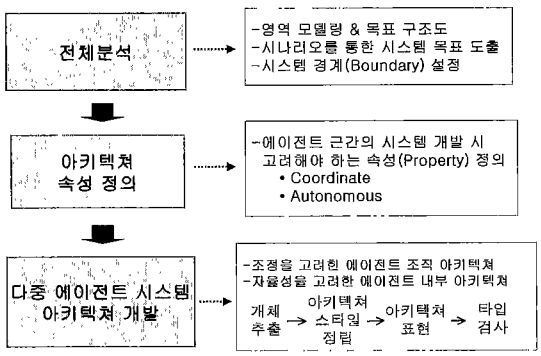

그림 1 다중 에이전트 시스템 개발을 지원하는 아키텍쳐 개발공정

시스템 전체 영역에 대한 분석을 통하여, 개발하는 시스템이 어떤 기능을 수행하는지를 파악한 후, 시스템 개발에서 고려해야하는 아키텍쳐 속성들을 정의한다. 이렇게 시스템의 전체 분석의 결과와 정의된 아키텍쳐 속성들을 바탕으로 실제 다중 에이전트 시스템의 아키텍쳐를 개발한다. 각 단계에 대해 살펴보면 다음과 같다.

3.1.1 전체분석

전체분석단계에서는 영역 모델링(Domain Modeling)을 통하여, 개발하는 영역에 필수적인 기능들을 분석하고, 목표 계층 시스템 다이어그램(Goal Hierarchy System Diagram)을 이용하여 영역 모델링에서 정의된 서비스들이 어떻게 표현되는지, 시스템의 기능이 모두 표현되고 있는지 확인해본다. 시스템의 목표는 시나리오에 기초하

여 시나리오와 목표의 집합을 구한 후, 정제과정을 통하여 시스템구축에 필수적인 목표들을 도출해낸다[10].

3.1.2 아키텍쳐 속성 정의

전체 분석을 통하여 시스템을 이해한 후, 시스템 개발에서 고려해야하는 아키텍쳐 속성들을 정의한다. 이러한 속성들은 개발하는 시스템 영역에 따라 그 중요도가 달라질 수 있는데, 분산된 환경에서 복잡한 문제를 효율적으로 처리하기 위한 다중 에이전트 시스템은 속도와 안정성, 확장성과 유연성의 장점을 가지고 있다. 이러한 아키텍쳐 속성들을 만족시키기 위하여 다중 에이전트 시스템은 조정과 자율성을 지원해야 한다. 특화된 기능을 수행하는 에이전트들간의 조정을 통하여 복잡한 일을 수행할 수 있는 구조를 가져야 하며, 각각의 에이전트는 변화하는 환경에서 자율적으로 작업을 처리할 수 있도록 유연한 구조로 구현되어야 한다.

3.1.3 다중 에이전트 시스템 아키텍쳐 개발

다중 에이전트 시스템 개발을 지원하는 아키텍쳐는 에이전트간 조정을 고려한 에이전트 조직 아키텍쳐와 에이전트의 자율성을 고려한 에이전트 내부 아키텍쳐로 나누어 볼 수 있다. 에이전트 조직 아키텍쳐는 시스템의 신뢰성을 고려하고, 상호작용을 유연하게 해주는 기능을 지원하여야 하고, 에이전트 내부 아키텍쳐는 동적인 환경에 자율적이고 능동적으로 대처할 수 있도록 지원하여야 한다. 각각의 관점마다 아키텍쳐 속성을 지원하기 위하여, 체계적인 공정이 필요하며 이는 개체추출, 아키텍쳐 스타일 정립, 아키텍쳐 표현, 타입 검사의 순서를 따른다. 3.2절에서는 조정과 자율성의 두 가지 관점에서 바라보는 아키텍쳐에 대해 공정별 산출물과 이의 도출과정을 좀 더 자세히 살펴보고자 한다.

3.2 다중 에이전트 시스템 개발을 지원하는 아키텍쳐

3.2.1 에이전트간 조정 관점에서의 아키텍쳐

다중 에이전트 시스템의 조정을 지원하는 아키텍쳐는 시스템 구성시 중요한 에이전트들을 정의하고 그들 사이의 관계를 표현한다. 조정은 개개의 에이전트들이 이루는 집단이 서로 화합하여 행동하게 하는 과정으로, 에이전트들 사이의 조직적 구조와 이들의 작업과 자원 할당에 대한 기준을 제시해 준다. 에이전트 조직 아키텍쳐의 도출공정과 단계별 산출물은 그림 2와 같다.

전체 분석을 통하여 얻어진 목표 계층 시스템 다이어그램과 목표 정제과정을 통하여 시스템의 목표들을 정의한다. 추출된 목표의 정제과정과 에이전트 매핑규칙을 적용하여 필요한 에이전트를 도출하고 각 에이전트의 역할을 정의한다. 도출된 에이전트들의 표기는 표 1과 같다.

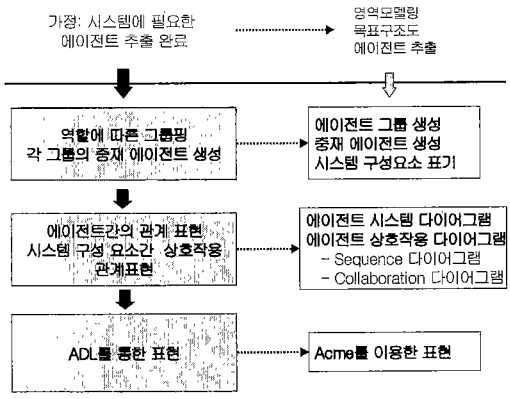

그림 2 조정을 지원하는 에이전트 조직 아키텍쳐 개발공정

표 1 에이전트 표기

Agent: A(G, {R}) G: 에이전트의 목표 {R}: 에이전트가 수행하는 역할들의 집합

도출된 에이전트들을 분석하여 시스템이 수행하는 역할들에 에이전트가 가지는 역할들을 매핑시켜 시스템의 역할 각각에 관여하는 에이전트 그룹을 형성한다. 각각의 에이전트는 자신이 수행할 수 있는 역할을 정의하고 있으므로, 공통의 역할을 가지는 에이전트끼리 그룹을 구성할 수 있게 되는데 에이전트 그룹은 표 2와 같이 표현될 수 있다.

표 2 에이전트 그룹 표기

Agent Group: Gr(R, {A}) R: 에이전트 그룹이 수행하는 공통의 역할 {A}: 역할 R에 관여하는 에이전트들의 집합

이 때, 하나의 역할은 여러 에이전트가 모여 수행하고, 하나의 에이전트는 시스템의 여러 역할을 수행하는데 관여할 수 있다. 이렇게 구성된 에이전트 그룹에는 그룹을 대표하여 그룹을 중재하는 중재자가 존재하여, 현재 시스템이 수행하는 역할을 위임받아 소속 에이전트들이 협력을 통하여 수행할 수 있도록 관리한다. 중재자는 표 3과 같이 표현된다.

표 3 에이전트 중재자 표기

Coordinator: C(R) R: 중재하고 있는 에이전트 그룹의 역할

위의 방법으로 다중 에이전트 시스템 개발에 필요한 주요 요소들(에이전트, 에이전트 그룹, 중재자)을 정의한 후, 이를 에이전트 시스템 다이어그램과 에이전트 상호작용 다이어그램으로 조직 아키텍쳐를 표현한다. 에이전트 시스템 다이어그램은 에이전트간의 상호작용을 시스템 전체관점에서 표현하는데, 그림 3은 에이전트, 에이전트 그룹, 중재자와의 매핑관계를 시스템 전체관점에서 표현한 일반적인 모습이다.

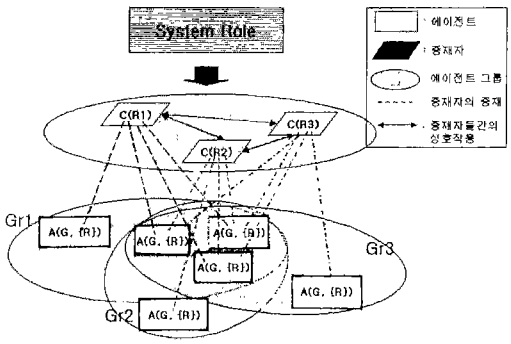

그림 3 에이전트 시스템 다이어그램의 일반적 형태

각 그룹간 에이전트 상호간의 협력은 개발하는 문제영역에서 가능한 시나리오를 수행할 때 에이전트들이 주고받는 메시지의 흐름을 표현함으로써 나타낸다. 다중 에이전트 시스템에서 많이 사용되는 조정 프로토콜은 Contract Net 프로토콜[11]을 들 수 있는데 이는 선창자(initiator)와 응답자(responder)사이의 관계를 통하여, 에이전트가 혼자 힘으로 문제를 해결하기 어려울 때 다른 에이전트들에게 작업을 공지하고 기다리면, 그 작업을 해결할 수 있는 에이전트가 비드(bid)를 보내게 되고 두 에이전트는 선창자와 응답자의 관계로 일을 처리하게 된다. 에이전트의 임무 변화는 다중 에이전트 시스템에서 서로 조정을 원할 때 표현되어진다. FIPA(the Foundation of Intelligent Physical Agents)[12]의 표준으로 지정되어 있는 이 규약은 Agent UML(AUML)을 적용시켜 표현할 수 있다[13]. AUML에 의하면, 다중 에이전트 사이의 조정은 Collaboration 다이어그램과 Sequence 다이어그램으로 표현할 수 있는데 Collaboration 다이어그램은 에이전트들 사이의 관계에 초점을 맞추는 반면, Sequence 다이어그램은 에이전트들 사이의 메시지 흐름에 초점을 맞춘다.

에이전트 시스템 다이어그램과 AUML의 Sequence 다이어그램과 Collaboration 다이어그램을 통한 조정을 지원하는 에이전트 조직 아키텍쳐는 ADL(Architecure Description Language)에 의해 시스템 전체의 구성요소와 그들간의 상호작용을 표현하고 그들간의 관계가 올바르게 설정되었는지 검사해 볼 수 있다. ADL은 디자인 단계에 보다 초점을 맞춰 시스템을 보여주기 때문에, UML에서 미약한 연결자(Connector)의 역할을 보다 명확하게 설명해 주며, ADL을 지원하는 도구를 사용하여 시스템 개발의 초기단계에 시스템의 구성요소들의 올바른 조합에 대해 검증해 볼 수 있다. 조정을 지원하는 아키텍쳐를 ADL의 하나인 Acme[14]로 표현해 보면 표 4와 같은 형태를 가진다. 표 4를 보면, 다중 에이전트 시스템을 구성하는 에이전트와 중재자는 컴포넌트로, 이들 사이의 통신규약은 커넥터로서 표현한다. 커넥터의 역할(Role)은 선창자와 응답자의 역할을 가지며, 이는 에이전트와 중재자에 정의된 통신포트와 연결된다.

표 4 Acme로 정형화한 조정을 지원하는 아키텍쳐

```
System MAS = {
    component Agent = {
            Port AgentComm;    };
    component Coordinator = {
            Port CoorComm;    };
    connector Communication = {
            Roles {      contractInitiator;
                         contractResponder;      };    };
    Attachments{
            Coordinator.CoorComm to
                         Communication.contractInitiator;
            Agent.AgentComm to

Communication.contractResponder;
            };          }
```

에이전트간 조정관점에서의 아키텍쳐를 개발함으로써 얻어지는 산출물과 그의 내용을 정리하면 표 5와 같다.

표 5 조정을 지원하는 아키텍쳐 관점에서의 산출물

산출물	내용
에이전트 시스템 다이어그램	에이전트, 에이전트 그룹, 중재자의 매핑관계 표현 에이전트들 사이의 상호작용관계를 시스템 전체관점에서 표현
에이전트 상호작용 다이어그램 - Collaboration 다이어그램 - Sequence 다이어그램	각 그룹의 에이전트간 상호협력을 주고받는 메시지의 흐름으로 표현 - 에이전트간 관계에 초점 - 에이전트들 사이의 메시지 흐름에 초점
ADL을 통한 정형화	Acme를 이용한 정형화

3.2.2 에이전트의 자율성 관점에서의 아키텍쳐

다중 에이전트 시스템의 자율성을 지원하는 아키텍쳐 도출공정과 각 단계별 산출물은 그림 4와 같다.

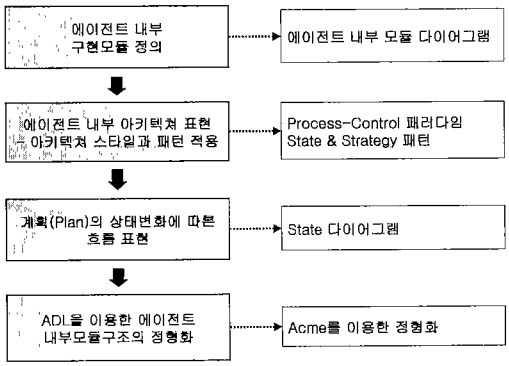

그림 4 자율성을 지원하는 에이전트 내부 아키텍쳐 공정

다중 에이전트 시스템을 구축하는데는 에이전트들간의 조정관점에서 아키텍쳐를 표현함으로써 시스템 전체의 구성을 파악하는 것과 더불어, 변화하는 환경에서 자율적으로 작업을 처리하는 에이전트 내부에 대한 아키텍쳐가 필요하다. 여기서 '자율적으로 작업을 처리한다'는 뜻은, 시스템을 구성하는 단위인 에이전트(일반적으로 하나의 컴포넌트)가 자신의 기능을 수행함에 있어서 다른 에이전트의 상태나 속성들에 영향을 받지 않음을 말한다. 즉, 에이전트는 자신이 가지는 기능을 상황에 따라 수행할 수도 있고, 외부로부터의 요청을 거부할 수도 있다. 이러한 에이전트의 자율성은 에이전트 내부의 아키텍쳐를 정의함에 있어 동적인 환경에 자율적이고 능동적으로 대처할 수 있는 제어능력을 가지도록 표현함으로써 달성될 수 있다.

자율성의 관점에서 에이전트의 내부 아키텍쳐를 정의하기 위해서는 에이전트 내부를 구현하기 위하여 필요한 모듈들을 정의하고 이들간의 관계를 정의하여야 한다. 에이전트의 내부는 목표(Goal), 자기정보(Belief), 계획(Plan), 그리고 기능(Capability)으로 구성된다[17]. 목표는 에이전트에게 주어지는 과업을 말하며, 이것은 바로 에이전트가 존재하는 이유가 된다[10]. 자기정보는 에이전트가 알고 있는 정보로서, 환경에 대한 정보 및 에이전트 자신에 대한 정보 등을 포함하며 계속적으로 갱신될 수 있다. 계획은 에이전트가 목표를 달성하기 위해 취하는 행동들을 나타낸 것으로, 에이전트는 자신의 목표와 자기정보를 참조하여 행동을 결정할 수 있다. 마지막으로, 기능은 에이전트가 자신의 계획에 따라 수행할 수 있는 작업들이다. 에이전트의 내부를 구현하기 위하여 필요한 모듈들은 계획모듈, 자기정보 모듈, 기능모듈이고, 기능모듈은 내부적으로 수행하는 기능들과 통신에 필요한 기능모듈로 나눌 수 있다. 에이전트의 내부속성과 구현모듈 사이의 관계는 그림 5와 같이 표현될 수 있다.

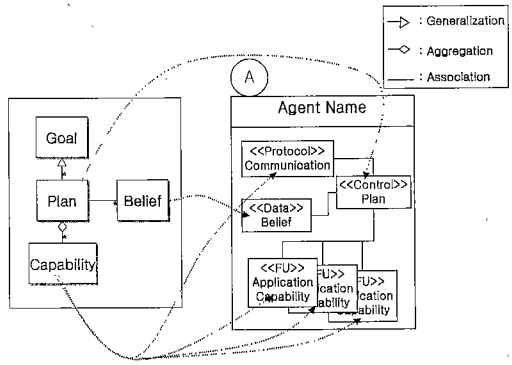

그림 5 에이전트의 내부속성과 구현모듈간의 관계

시스템에서 특정 역할을 하는 에이전트는 정해진 목표를 성취하며, 그 목표는 자신의 일반화된 형태인 계획의 구현으로 달성된다. 또한 계획은 그 과정에서 자기정보를 사용하게 되며, 계획의 일반화된 형태인 기능들의 조합과 통신수단을 이용한 다른 에이전트와의 협력을 통하여 작업을 수행할 수 있다. 이는 자기정보 모듈과 통신기능모듈, 내부실행기능 모듈의 구현을 통하여 가능하다.

에이전트 내부구현에 있어, 아키텍쳐의 중추적 역할을 담당하는 것은 계획모듈이라고 할 수 있다. 계획모듈은 자기정보 모듈을 참고하여 현재상황을 인식하며, 이에 적합한 기능을 선택하여 수행하게 하는 제어의 역할을 한다. 이를 아키텍쳐 스타일로 표현했을 때 Process-Control 패러다임에 적용할 수 있다. 이 스타일은 소프트웨어를 제어하는 부분과 프로세스를 담당하는 부분을 분리하여 아키텍쳐를 구성하는 것으로, 입력이 들어왔을 때, 제어기능을 담당하는 부분이 목표치(Set Point)와 비교하여 행해야 하는 기능을 결정하고 이를 프로세스 부분에 전달한다. 이러한 과정은 목표치가 달성될 때까지 앞의 과정을 계속한다.

Process-Control 패러다임은 에이전트와 같이 자율성을 가지고 자신의 목표를 성취하기 위해 상황에 따라 적절한 기능을 수행하는 소프트웨어 아키텍쳐를 정의하는데 적합하다. 이를 에이전트의 구현모듈에 적용하면 그림 6과 같다.

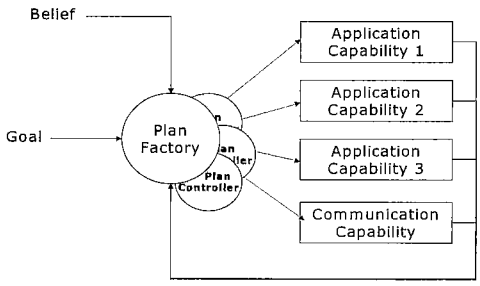

그림 6 에이전트 내부구현모듈들의 관계에 적용한 Process-Control 아키텍쳐 스타일

하나의 에이전트는 자신의 기능을 수행하기에 앞서, 목표와 자기정보를 가지고 현재의 상태를 파악하게 되고, 그 상태에 따라 적절한 기능을 선택적으로 수행하게 된다. 계획모듈은 제어의 역할을 담당하며, 자신의 목표가 달성될 때까지 자기정보모듈을 참조하여 그때 그때의 상황에 맞는 내부실행기능 모듈을 선택하는 전략을 갖는다. 목표를 성취하기 위해 가지는 하나 이상의 계획들은 에이전트의 역할과 매핑될 수 있으며, 자신의 역할을 수행함으로써 여러 방법으로 목표를 달성할 수 있다. 계획모듈은 Plan Factory라는 집합으로 표현될 수 있으며, 수행하는 역할에 따라 그에 가능한 계획을 선택하게 된다.

에이전트가 특정 상태에서 수행되는 전략을 진행할 것인지의 여부를 결정하는 방법은 이전 수행기능의 결과와 현재의 자기정보에 따라 결정될 수 있는데 이는 표 6과 같이 나타낼 수 있다.

표 6 수행할 기능을 선택하는 조건

∃ Capability(satisfied) such that current belief ∩ post-cond of Pre_capability ⊆ pre-cond   ⇒ 이전에 수행된 기능의 결과와 현재의 자기정보를 고려한 상태가 자신의 선행조건을 만족시키는 기능

에이전트가 자신의 계획과 기능을 상황에 따라 선택하는 것은 여러 가지 패턴[15]들을 사용하여 구현할 수 있으며, State 패턴과 Strategy 패턴을 사용하면 내부상태의 변화에 따라 동적으로 기능을 선택할 수 있다.

앞에서 언급한 것과 같이, 계획은 목표가 달성될 때까지 자기정보를 참조하여 그때 그때의 상황에 맞는 기능을 선택한다. 이러한 상태를 표현하는 것은 에이전트의 계획을 UML의 State 다이어그램으로 나타냄으로써 가능하다. 즉, 에이전트가 도달할 수 있는 상태에 따라 수행하는 계획과 기능을 정의함으로써 에이전트가 목표를 달성하는 과정을 표현할 수 있다.

시스템의 전체관점에서 조정을 지원하는 아키텍쳐를 아키텍쳐 표현언어인 Acme를 이용하여 정형화하고 구조를 검사한 것처럼, 에이전트의 내부 구현모듈 아키텍쳐 또한 Acme를 이용하여 정형화시킬 수 있다. 자기정보모듈과 통신기능모듈, 내부수행기능모듈, 그리고 계획모듈의 기본 형태를 에이전트와 중재자 각각에 'Family' 타입으로 정의해 놓은 후 이를 실제 영역에 적용시킬 때 상속하여 표현하는 방법(AcmeFamily)을 사용한다. 에이전트의 내부 구현모듈 아키텍쳐를 Acme로 정형화하면 표 7과 같다.

표 7 Acme로 정형화한 에이전트 내부 아키텍쳐

```
Family Agent = {
    Component Type Belief = {
        Port btoPlan;
        Property AgentInfo : record[];
        Property DataInfo : record[];  };
    Component Type ApplicationCapability = {
        Port ctoPlan;
        Property Precondition : boolean;
        Property Postcondition : boolean;          };
    Component Type CommunicationCapability = {
        Port ctoPlan; Port ctoComm;
        Property Precondition : boolean;
        Property Postcondition : boolean;          };
    Connector Type Plan = {
        Role { roleForState1; roleForState2;
               roleForState3; roleForState4; }
        Property Goal : string;
        Property stateList =
        <"register", "checkBelief", "checkCapability",
         "communication">;
    };
};
```

에이전트의 내부 구현모듈은 자기정보모듈과 통신 및 내부 수행 기능모듈을 컴포넌트로, 계획모듈을 커넥터로 나타낼 수 있다. 계획모듈은 에이전트 내부의 제어역할을 담당하는 것으로 상황에 따라 자기정보 모듈을 참조하여 적절한 기능을 선택한다. 따라서, 자기정보 모듈과 기능 모듈들에 정의된 포트들은 계획모듈에서 정의한 역할에 매핑되며 이는 실제 시스템의 에이전트를 나타낼 때 'Attachments'하에 정의된다. 에이전트의 내부는 시스템 전체를 나타낼 때, 에이전트와 중재자 각각의 컴포넌트를 'Representation' 방식을 이용하여 에이전트 내부의 구현모듈을 자세히 나타냄으로써 표현할 수 있으며, 외부 통신을 위해 정의된 Port는 통신기능을 담당하는 모듈에서 지정한 Port와 연결된다.

다중 에이전트 시스템의 자율성을 지원하는 관점에서의 아키텍쳐를 개발할 때 얻어지는 산출물과 그의 내용을 정리하면 표 8과 같다.

표 8 자율성을 지원하는 아키텍쳐 관점에서의 산출물

산출물	내용
에이전트 내부모듈 구조도(Agent Internal Module 다이어그램)	에이전트 구현을 위한 내부 모듈간의 관계표현
스타일과 패턴을 이용한 에이전트 내부 아키텍쳐 표현	아키텍쳐 스타일과 패턴의 적용 - Control-Process 패러다임 - State & Strategy 패턴
에이전트 내부 상태도	에이전트가 목표를 달성하는 과정 에이전트의 내부 속성 중 제어기능을 담당하는 계획(Plan)부분의 표현 - State 다이어그램
ADL을 통한 표현	Acme를 이용한 표현

4. 출발전 교통정보 안내 서브시스템(PTIS) 예제의 적용

본 장에서는 3장에서 제안한 다중 에이전트 시스템 개발을 지원하는 아키텍쳐 개발방법을 검증의 한 과정으로, 지능형 교통정보 시스템의 출발전 교통정보 안내 서브시스템에 적용해보고 이에 대한 프로토타입을 개발해 본다.

4.1 문제영역

4.1.1 시나리오

적용하는 출발전 교통정보 안내 서브시스템의 시나리오는 표 9와 같다.

표 9 출발전 교통정보 제공 서브 시스템의 시나리오

사용자는 12월 10일부터 20일까지 제주도 여행을 하고자 한다. 여행하기에 앞서, 제주도까지의 비행기편과 여행기간동안의 날씨에 대해 알아보기 위하여 사용자는 '출발 전 교통정보 안내' 사이트에 접속을 시도한다. 서비스를 시작하자, ID와 비밀번호를 입력하라는 요청을 받는다. 아이디 'SELAB', 패스워드 'SSEL1008'을 입력하자 교통정보제공을 위한 기본화면이 출력된다. 사용자는 출발지, 도착지, 출발일, 도착일을 입력한다. '출발 전 교통정보 서비스' 시스템은 사용자에게 항공사의 비행시간, 가능한 좌석 수 등을 표시해준다. 예약을 원하면 그 사이트로 이동하도록 한다. 사용자는 여행기간동안의 날씨를 알고자, 주간날씨를 제공하는 화면으로 이동하여 주간날씨를 제공받는다. 여행 출발 당일, 사용자는 공항까지의 교통편과 현재 교통상황을 알아보기 위하여, '출발 전 교통정보 서비스' 사이트에 접속하여 현재 도로상황과 공항까지의 교통수단(지하철노선), 공항까지의 소요시간을 확인한다.

출발전 교통정보 제공 서브시스템은 본래, 지능형 교통시스템의 PTIS(Pre-trip Traveler Information Systems) 시나리오[3]의 일부로, 공중단말장치나 개인단말장치를 사용하는 사용자에게 교통·업무·여행관련의 모든 편의정보 및 여행경로를 제공하는 전체 시나리오에서 항공스케줄정보와 기상정보, 도로정보, 그리고 지하철정보를 제공하는 것으로 한정시켰다.

4.2 아키텍쳐의 적용

4.2.1 목표 계층 시스템 다이어그램

출발전 교통정보 제공 서브 시스템의 시나리오를 분석하여 도출된 목표 계층 시스템 다이어그램은 그림 7와 같다.

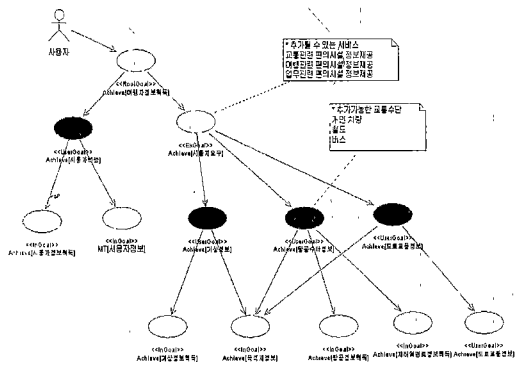

그림 7 문제영역에 적용한 목표 계층 시스템 다이어그램

목표 계층 시스템 다이어그램을 바탕으로 추출된 목표들의 정제과정을 통해 매핑된 에이전트는 모두 여섯이며, 이는 사용자 상호작용 에이전트, DB 랩퍼 에이전트, 항공스케줄제공 에이전트, 고속도로정보 에이전트, 지하철정보 에이전트, 그리고 기상정보 에이전트이다. 추출한 에이전트들을 바탕으로 다중 에이전트 시스템 개발을 지원하는 아키텍쳐를 조정을 고려한 에이전트 조직 아키텍쳐와 에이전트의 자율성을 고려한 에이전트 내부 아키텍쳐를 살펴보면 다음과 같다.

4.2.2 에이전트간 조정 관점에서의 아키텍쳐

조정을 지원하는 아키텍쳐를 개발하기 위해서는 전체 분석을 통하여 얻어진 목표들을 정의하고, 추출된 목표의 정제과정과 에이전트 매핑규칙을 적용하여 필요한 에이전트를 도출한다. 출발전 교통정보 제공 서브시스템에 적용한 에이전트와 에이전트 그룹, 중재자를 표현하면 그림 8과 같다.

•**Agents – A(G, {R})**
- User Interface A: A (외부와 상호작용, R{(항공수단제공), (도로정보제공), (기상정보제공)})
- Flight Info A: A (항공스케줄 제공, R{(항공수단제공)})
- Subway Info A: A (지하철경로제공, R{(항공수단제공), (도로정보제공)})
- Weather Info A: A (기상정보제공, R{(기상정보제공)})
- Highway Info A: A (고속도로정보제공, R{(도로정보제공)})
- DB Wrapper A: A (DB관리, R{(항공수단제공)})

•**Agent Groups – Gr(R, {A})**
- Gr (항공수단제공, {User Interface, Flight Info, Subway Info, DB Wrapper})
- Gr (도로정보제공, {User Interface, Highway Info, Subway Info})
- Gr (기상정보제공, {User Interface, Weather Info})

•**Coordinators – C(R)**
- C (항공수단정보중재)
- C (도로정보중재)
- C (기상정보중재)

그림 8 전체분석을 통해 도출된 에이전트, 에이전트 그룹, 중재자

출발전 교통정보 제공 서비스를 제공하기 위해서는 외부와 상호작용하는 에이전트가 있어서 사용자의 요구사항을 받아들이고 원하는 결과를 보여주는 역할을 담당한다. 항공스케줄 제공 에이전트는 다양한 항공사의 원격 데이타베이스에 접근하여 출발지와 도착지의 날짜별 항공 데이타를 얻어와 이를 표준 포맷으로 통일하는 작업을 담당하며, 지하철 정보 에이전트와 고속도로정보 에이전트, 기상정보 에이전트는 각각 사용자가 원하는 위치의 도로정보와 지하철경로정보, 기상정보를 가지고 온다. DB 랩퍼 에이전트는 데이타베이스를 마치 에이전트처럼 보이게 하여 정보를 요청하는 일관된 포맷을 정의하고 이에 맞는 정보를 제공해주는 역할을 한다.

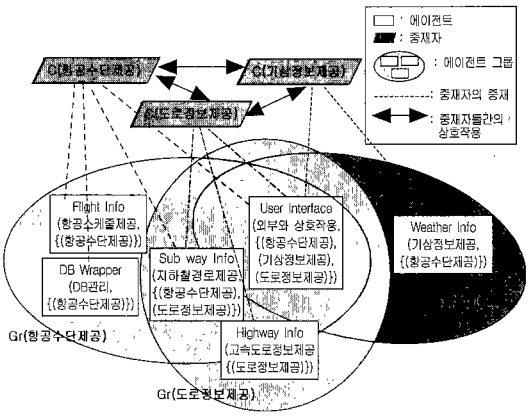

그림 9 출발전 교통정보 제공 서브시스템 영역에 적용하여 도출한 에이전트 시스템 다이어그램의 일반적 형태

도출된 에이전트간의 역할에 따라 그룹핑 할 수 있으며 여기서 그룹의 에이전트들을 관리하는 중재자를 생성한다. 출발전 교통정보 제공 서비스를 실현함에 있어서는 세 개의 그룹이 구성되는데, 이는 기상정보제공그룹, 도로정보제공그룹, 항공수단제공그룹이다. 각 그룹에는 중재자가 있어서 시스템이 수행하는 역할에 따라 각 그룹의 에이전트들이 활성화되어 작업을 수행하게 된다. 그림 9는 출발전 교통정보 제공 서브시스템 영역에 적용하여 도출한 에이전트 시스템 다이어그램의 일반적인 모습이고, 그림 10는 이 중 항공수단정보를 제공하는 그룹의 에이전트와 중재자간의 관계를 UML 표기로 표현한 것이다.

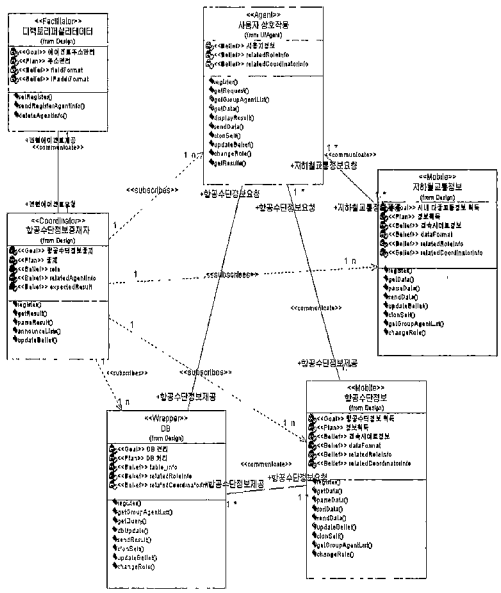

그림 10 항공수단제공을 담당하는 그룹의 UML 표현

그림 10을 살펴보면, 모든 에이전트는 지원하는 기반구조의 퍼실리테이터(Facilitator)에게 등록한 후 서로 상호작용하게 된다. 중재자는 그룹 에이전트들에 대한 정보를 가지고 있어서, 사용자의 요청이 들어오면 자기 정보를 참조하여 그룹에 속하는 에이전트들에게 에이전트 리스트 메시지를 보내고, 각각의 그룹 에이전트들은 상호작용 메시지를 보냄으로써, 작업을 수행하게 된다.

그림 11은 그림 10에서 표현한 항공수단제공 그룹의 구성에이전트간의 상호작용을 메시지의 흐름으로 표현한 Sequence 다이어그램이다. 각 에이전트는 Contract Net 프로토콜을 따르며, 선창자와 응답자로서의 임무를 수행할 수 있다.

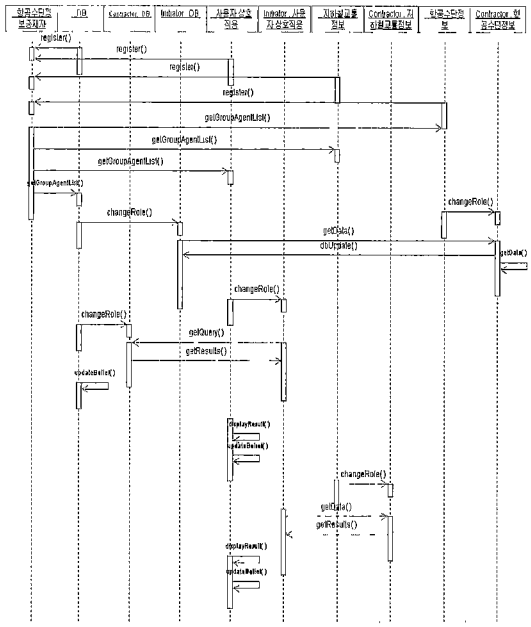

그림 11 그림 10에 나타난 에이전트들의 상호작용을 표현한 AUML의 Sequence 다이어그램

에이전트가 선창자로서의 임무를 가질 때는 다른 에이전트들에게 자신이 필요한 데이타에 대한 요청을 하게 된다. 이 요청을 받은 에이전트들은 자신이 수행 가능한 작업일 경우, 응답자로서의 임무를 수행하여 요청한 데이타나 기능에 대한 결과를 보내줄 수 있다. 그림 11에서 사용자 상호작용 에이전트는 항공수단정보를 가져올 때, 선창자로서 그룹의 다른 에이전트들에게 요청을 하게 되면, DB 랩퍼 에이전트는 응답자로서 결과를 보내주게 된다. 또한, DB 랩퍼 에이전트는 항공수단정보를 얻기 위하여 그룹의 다른 에이전트들에게 선창자로서 정보를 요청하게 되며, 이 때 항공수단정보 에이전트는 응답자로서 결과를 보내주게 된다.

4.2.3 에이전트의 자율성 관점에서의 아키텍처

자율성을 지원하는 아키텍쳐를 개발하기 위해서는 에이전트 내부구현을 위해 필요한 모듈들과 이들간의 관계에 대해 정의하여야 한다. 그림 5에서 제시한 것과 같이 에이전트는 하나의 목표를 가지며, 그 목표에 따라하나 이상의 계획을 가지게 된다. 계획모듈은 자기정보모듈을 참조하고, 자신의 계획을 성취할 수 있는 여러 기능모듈들로 이루어져 있다. 출발전 교통정보 제공 서브시스템에서 도출된 에이전트 중, 사용자 상호작용 에이전트에 적용하여 내부구현모듈을 표현하면 그림 12와 같다.

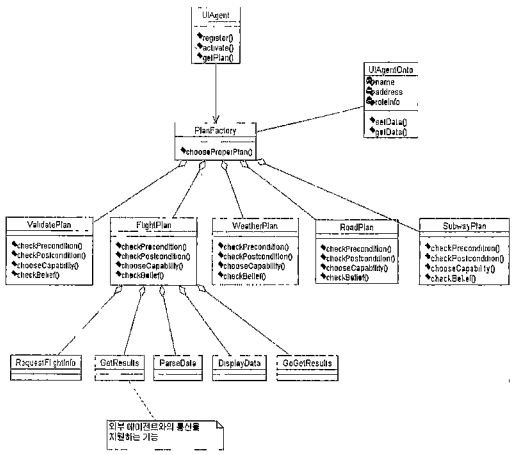

그림 12 UML에 의한 사용자 상호작용 에이전트의 내부구현모듈 표현

문제영역의 사용자 상호작용 에이전트는 생성시, 퍼실리테이터에게 자신을 등록하며 이 때 자신이 가지는 역할에 따라 중재자의 에이전트 리스트에 등록되게 된다. 사용자의 요청이 들어오면, 자신의 계획 모듈을 살펴보고 적절한 계획을 선택하게 되며, 각각의 계획모듈은 이를 만족시킬 수 있는 기능모듈들을 가지고 있어서 자기정보의 상황에 따라 적절한 기능을 선택하게 된다. 아키텍쳐 스타일의 하나인 Process-Control 패러다임으로 사용자 상호작용 에이전트의 내부모듈을 표현하면 그림 13과 같다.

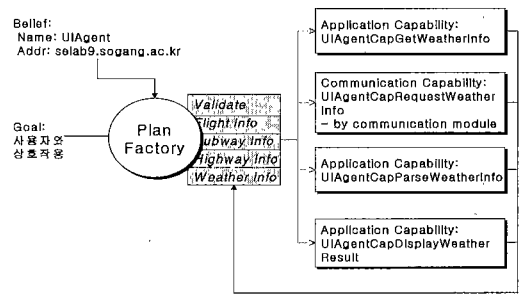

그림 13 Process-Control 패러다임을 적용한 사용자 상호작용 에이전트의 내부 아키텍쳐

그림 13에서 보는바와 같이, 사용자 상호작용 에이전트의 계획모듈과 이를 구성하는 다양한 계획들은 제어의 역할을 담당하며 각 계획마다 적절한 기능을 선택하

게 되는데 이때 고려하는 변수가 자기정보와 '사용자와 상호작용을 달성한다' 라는 목표이다.

'사용자와 상호작용을 달성한다'라는 목표를 달성하기 위하여 사용자 상호작용 에이전트는 자신의 변화하는 상태에 따라 여러 가지 전략을 가지고 목표를 달성하는데 이는 State 다이어그램을 통하여 표현할 수 있다. 그림 14는 사용자 상호작용 에이전트가 목표를 달성하기 위하여 보이는 상태들을 표현한 State 다이어그램이다. 이는 사용자 상호작용 에이전트가 가질 수 있는 상태를 크게 바라본 것으로, 각 상태는 세부 상태들의 흐름으로 자세히 표현 할 수 있다.

에이전트가 처음 생성되면, 가장 먼저 수행하는 일이 자신이 속한 그룹의 중재자에게 등록을 하는 것이다. 자기정보에 자신이 가지는 역할을 정의하고 있어서, 그 역할에 따라, 원하는 그룹의 중재자에게 등록을 한다. 이렇듯, 에이전트는 자신의 목표를 달성하기 위하여 하나 이상의 계획을 가지고 있으며, 각각의 계획은 에이전트가 도달하는 상태에 따라 적절한 기능모듈을 결정하게 된다.

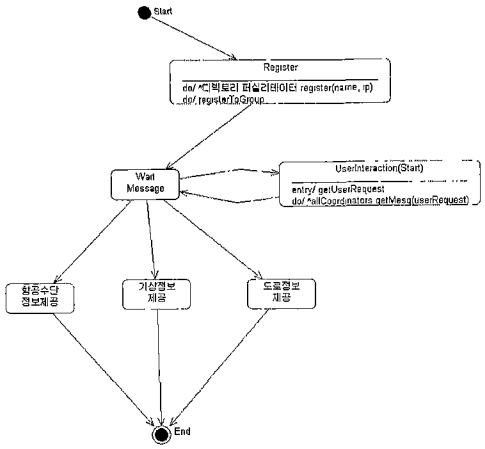

그림 14 사용자 상호작용 에이전트가 목표를 수행하기 위해 도달하는 상태를 표현한 State 다이어그램

4.2.4 Acme로 표현한 문제영역 아키텍처

출발전 교통정보 제공 서브 시스템 개발을 위하여 제안한 아키텍처를 적용하여 조정의 관점과 자율성의 관점에서 표현하였다. 아키텍처 스타일과 패턴, UML을 이용하여 표현한 두 가지 관점의 아키텍처를 통합하여 Acme를 이용하여 표현할 수 있다. 그림 15는 시스템을 구성하는 컴포넌트와 커넥터를 정의한 것으로, 시스템을 구성하는 에이전트와 중재자는 컴포넌트로, 이들 사이의 통신 프로토콜은 커넥터로서 정의하였다. Acme의 'Family' 타입은 시스템의 요소들에게 요구되는 기본구조를 정의해 놓은 것으로서, 시스템 전체의 입장에서 컴포넌트로 선언되어 있는 에이전트와 중재자는 자기들만의 기본구조를 가지고 있다.

에이전트와 중재자는 모두 에이전트로서, 이들의 내부 구현모듈은 그림 5에서 정의한 바와 같이 자기정보 모듈, 계획 모듈, 통신기능 모듈, 그리고 내부의 수행기능 모듈이다. 에이전트와 중재자의 자기정보 모듈과 수행기능모듈, 통신모듈들은 각각 컴포넌트로 선언할 수 있으며, 이들은 에이전트의 계획모듈과 연결되어 있어 선택되어지므로, 계획을 커넥터로 선언한다. 계획의 속성으로 목표를 정의해 놓아, 다음 기능을 수행하는데 항상 참조할 수 있도록 하였다. 그림 15는 시스템을 구성하는 컴포넌트와 커넥터의 기본구조를 보여주는 것이다. 이와 같이, ADL을 이용하여 시스템을 표현함으로써 시스템 전체의 중요요소들과 그들 사이의 관계를 명확히 설명할 수 있으며, 이를 검증해 봄으로써, 개발의 초기단계에 올바른 시스템 개발을 위한 아키텍처를 지원할 수 있다.

```
Family Agent = {
    Component Type Belief = {
        Port btoPlan;
        Property AgentInfo : record[];
        Property DataInfo : record[];   };
    Component Type ApplicationCapability = {
        Port ctoPlan;
        Property Precondition : boolean; Property Postcondition : boolean;   };
    Component Type CommunicationCapability = {
        Port ctoPlan;
        Port ctoComm;
        Property Precondition : boolean; Property Postcondition : boolean;   };
    Connector Type Plan = {
        Role roleForState1; Role roleForState2; Role roleForState3;
        Role roleForState4;
        Property Goal : string;
        Property stateList =
            <"register", "checkBelief", "checkCapability", "comm">;
        Property strategyList = <"strategy1", "strategy2">;   };
};

Family Coordinator = {
    Component Type Belief = {
        Port btoPlan;
        Property CoordinatorInfo : record[]; Property AgentInfo : record[];   };
    Component Type ApplicationCapability = {
        Port ctoPlan;
        Property Precondition : boolean; Property Postcondition : boolean;   };
    Component Type CommunicationCapability = {
        Port ctoComm;
        Property Precondition : boolean; Property Postcondition : boolean;   };
    Connector Type Plan = {
        Role roleForState1; Role roleForState2; Role roleForState3;
        Property Goal : string;
        Property stateList = <"register", "monitor", "comm">;
        Property strategyList = <"strategy1", "strategy2">;   };
};

Connector Type Communication = {
    Role contractInitiator;
    Role contractResponder;
};
```

그림 15 Acme를 이용하여 표현한 시스템 구성요소의 기본구조

4.3 프로토타입의 구현

4.1절의 출발전 교통정보 제공 서브 시스템 시나리오를 적용하여 제안한 4.2의 아키텍처를 기반으로, 본 절에서는 프로토타입을 개발하여 에이전트 시스템 개발의 방향을 제시한다.

다중 에이전트 시스템을 위의 시나리오에 적용하여 프로토타입을 구현하기 위하여 사용되는 환경은 표 10과 같다.

표 10 프로토타입의 구현환경

기반구조를 지원하는 환경	JADE 2.01
DBMS	ORACLE 8i
운영체제	Linux(1대) Windows 98 ME(2대) Windows 2000(2대)
구현언어	JAVA
에이전트 통신언어	FIPA ACL 표준

JADE(Java Agent DEvelopment Framework)[16]는 자바언어로 구현된 에이전트 소프트웨어 개발을 지원하는 환경으로서, FIPA(the Foundation of Intelligent Physical Agents)에서 제정한 표준명세를 기반으로 분산된 환경에서 에이전트를 개발하고 이들간의 통신과 이동을 지원한다. JADE를 사용하여 다중 에이전트 시스템 개발을 위한 환경을 정의하고, 자바언어로 구현된 에이전트들은 JADE상에서 서로 조정을 통하여 사용자의 요구사항을 만족시킨다. 이때, 에이전트간의 통신은 FIPA에서 제정한 ACL 표준을 따르며, Contract Net 프로토콜의 개념을 사용하여 통신한다.

JADE에서 제공하는 주요 API들을 간략히 살펴보면 그림 16과 같다.

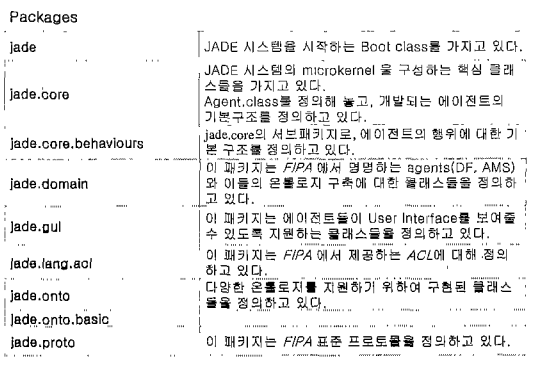

그림 16 JADE 2.01에서 제공하는 주요 API들

출발전 교통정보 제공 서브시스템의 프로토타입 개발을 위하여, JADE에서 제공하는 API를 이용, 또는 응용하였다. 문제영역에서 도출된 에이전트들과 중재자들은 모두 jade.core에 정의되어 있는 Agent 클래스를 확장하여 구현하였으며(단, 사용자 상호작용 에이전트는 jade.gui에 정의되어 있는 GuiAgent 클래스를 확장하여 구현), 에이전트의 내부 속성 중, 자기정보는 jade.onto에 정의되어 있는 Ontology 클래스를 이용하였다. 에이전트의 계획과 기능은 jade.core.behaviours에 정의되어 있는 Behaviour 클래스를 변형하여 구현하였는데, 계획은 기능들의 조합을 수행하는 행위로 보아, 기능들을 모아놓은 기능 리스트(CapabilityList)를 가지고 있어서, 자기정보에 따라 기능을 선택하도록 하였다. 이를 나타내면 그림 17과 같다.

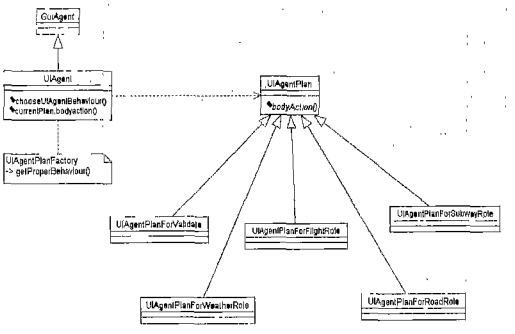

그림 17 사용자 상호작용 에이전트의 Plan 선택방법

출발전 교통정보 제공 서브 시스템이 제공하는 서비스 중 항공스케줄을 알려주는 부분에 대한 프로토타입을 보이면 그림 18과 같고, 이는 JADE가 지원하는 기

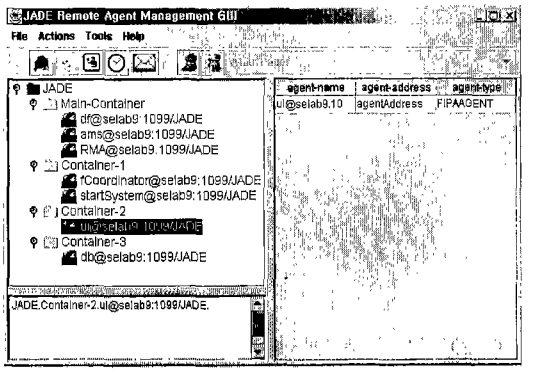

그림 18 JADE 기반구조에 등록된 에이전트 현황

반구조에 에이전트가 등록되었을 때의 모습을 나타내며, 이는 각 에이전트가 자신을 시작할 때, 가장 먼저 행하는 기능이다.

각 에이전트는 JADE 기반구조에 자신을 등록함과 동시에, 자신이 속한 그룹의 중재자에게도 등록을 하게 된다. 그림 19는 사용자 상호작용 에이전트가 그룹의 중재자들에게 자신을 등록시키는 통신메시지를 보여주며, 이는 ACL 포맷을 따른다.

그림 19 사용자 상호작용 에이전트가 중재자에게 등록하기 위해 보내는 통신 메시지

시스템은 사용자 인증이 끝나면, 서비스를 시작한다. 그림 20은 사용자 인증을 담당하는 인터페이스와 항공스케줄을 선택하는 인터페이스를 보여주며, 그림 21은 시나리오에 의거해 12월 10일 제주행 항공기의 스케줄을 보여주는 화면이다.

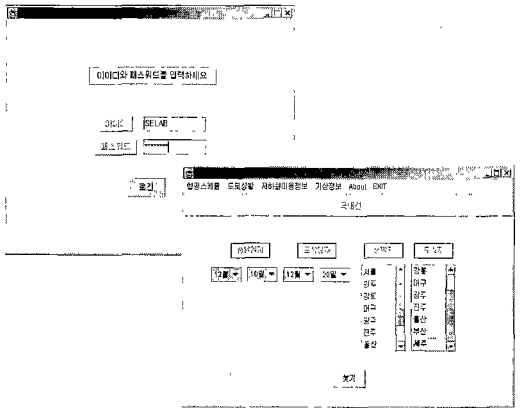

그림 20 사용자 등록과 항공스케줄 요청에 관한 User Interface

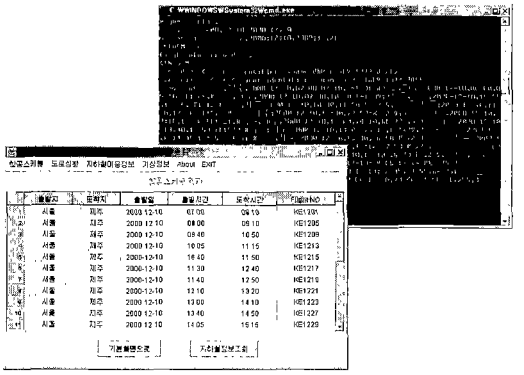

그림 21 항공스케줄 획득 메시지 및 이의 사용자 상호작용 에이전트 출력 결과

5. 결 론

실세계에서 발생하는 해결하기 어려운 복잡한 문제들을 해결하기 위한 노력으로, 다중 에이전트 시스템(Multi-Agent System) 구축에 대한 관심이 높아지고 있다. 다양한 종류의 분산 인공지능 문제들을 에이전트라는 추상적 단위와 에이전트간의 상호작용을 토대로 해결하는 시스템을 개발하기 위하여, 본 연구에서는 다중 에이전트 지향의 소프트웨어를 개발하는데 핵심이 되는 아키텍처 개발방법을 조정과 자율성의 관점에서 스타일과 패턴, 여러 표기법(UML과 ADL)을 이용하여 제안하였다. 문제영역을 분석하여, 다중 에이전트 시스템의 특성을 파악하고 그 특성들을 시스템 개발에 반영하기 위하여 필수적인 아키텍처 개발공정을 제안하였으며, 이를 조정을 지원하는 시스템조직 관점과 자율성을 지원하는 에이전트 내부 관점에서 표현함으로써, 복잡한 시스템의 외부와 내부를 쉽게 이해할 수 있도록 하였다. 또한, 제안한 아키텍처 개발방법을 적용한 프로토타입 개발을 통하여 에이전트 시스템 개발의 방향을 제시하였다.

분산된 환경에서 소프트웨어들이 협력하여 복잡한 서비스를 제공해 주기 위하여, 조정과 자율성을 기본으로 하는 에이전트를 기본단위로 소프트웨어를 개발하는 노력이 필요하다. 이를 지원하기 위하여 에이전트 근간의 소프트웨어를 개발하기 위한 방법론을 체계화하여야 하며, 본 연구는 개발의 핵심인 아키텍처를 제안함으로써, 소프트웨어의 골격을 바로 잡는 역할을 할 수 있다. 향후 연구과제로 본 논문에서 제안한 아키텍처 개발방법을 체계적으로 검증할 수 있는 검증방법이 필요하며, 이

를 여러 분야에 적용해 봄으로써 보다 논리적인 뒷받침을 제공해야 한다. 또한, 조정과 자율성을 지원하는 아키텍처와 더불어, 분산환경에서 이용되는 소프트웨어들의 보안성을 고려한 아키텍처의 연구가 필요하며, 에이전트 근간의 소프트웨어에서 에이전트의 속성 중 하나인 이동성(mobility)을 지원하는 아키텍처에 대한 연구 또한 진행되어져야 한다. 앞으로 연구될 보안과 이동성을 지원하는 아키텍처는 본 연구에서 제안한 두 가지 관점의 아키텍처와 함께, 에이전트 근간의 소프트웨어 개발방법론을 다양한 관점의 아키텍처 중심으로 풀어나가는 기반이 될 것이다. 본 연구의 결과로 그림 22와 같은 소프트웨어 개발의 방향을 제시한다.

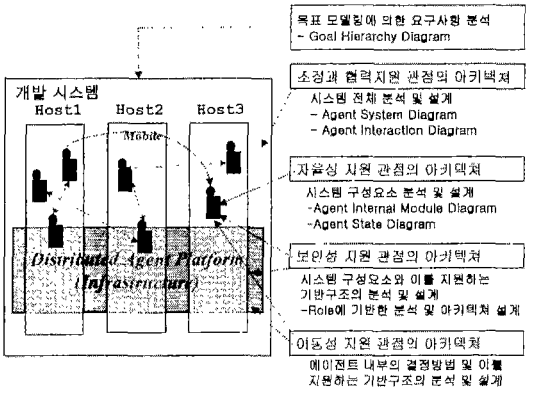

그림 22 아키텍쳐 근간의 다중 에이전트 시스템 소프트웨어 개발 방향

그림 22에서 제안하는 아키텍쳐를 바탕으로, 다양한 관점에서 시스템을 바라보고 아키텍쳐를 설계하여 이를 근간으로 소프트웨어 시스템을 체계적으로 개발할 수 있는 방법론이 개발되어야 한다. 이를 위하여, 각 관점간의 연관관계를 정의하고 규명하는 것을 지원하고, 이를 검증할 수 있는 방안의 연구와 함께, 개발하는 방법론을 지원하는 도구개발에 대한 연구가 필요하다.

참 고 문 헌

[1] Martin Fowler and Kendall Scott, "UML Distilled 2nd Edition," Addison Wesley, 2000.
[2] P.C. Clements, "A Survey of Architecture Description Languages," Proceedings of 8th Int'l Workshop Software Specification and Design, Mar, 1996.
[3] 건설교통부, "국가 ITS 아키텍쳐 확립을 위한 연구(II)", 국토연구원, 1999년 12월.
[4] OHare, G. and Jennings, N. (Eds), "Foundations of Distributed Artificial Intelligence," John Wiley & Sons, 1996.
[5] S. Green, L. Hurst, B. Nangle, P. Cunningham, F. Somers, and R. Evans, "Software Agents: A review," 1997.
[6] Hyacinth Nwana, Lyndon Lee, and Nick Jennings, "Coordination in Software Agent Systems," BT *Technology Journal, 14(4), 1996, pp 79-88.*
[7] David L. Martin, Adam J. Cheyer, and Douglas B. Moran, "The open agent architecture: A framework for building distributed software systems," Applied Artificial Intelligence, January-March 1999, Vol. 13, pp. 91-128.
[8] K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng. "Distributed intelligent agents", IEEE Expert-Intelligent Systems and Their Applications, 1996, 11(6), pp. 36-45.
[9] Frances Brazier, Barbara Dunin-Keplicz, Nick R. Jennings, and Jan Treur. "Formal specification of multi-agent systems: a real-world case," In Victor Lesser, editor, Proceedings of the First International Conference on Multi-Agent Systems, San Francisco, CA, 1995, MIT Press, pp 25-32.
[10] Kenha Park, Jintae Kim, and Sooyong Park, *"Goal-based agent-oriented software modeling,"* APSEC, Dec, 2000.
[11] Davis R and Smith R. G, "Negotiation as a metaphor for distributed problem solving," Artificial Intelligence 1983, pp. 63-109.
[12] Foundation for Intelligent Physical Agents. Specifications. 1999. http://www.fipa.org.
[13] James Odell, H. Van Dyke Parunak, Bernhard Bauer, "Representing Agent Interaction Protocols in UML," AAAI Agents 2000 conference, Barcelona June 2000, 1999.
[14] D. Garlan, R. Monroe, and D. Wile, "Acme: An Architecture Description Interchange Language," Proc. of CASCON'97, Nov 1997.
[15] Erich G., Richard H., Ralph J., and John V., "Design patterns: Elements of Reusable Object-Oriented Software," Addison- Wesley, 1995.
[16] *Fabio Bellifemine, Agostino Poggi, and Giovanni* Rimassa, "JADE-A FIPA-compliant agent framework," Proceedings of PAAM'99, London, April 1999, pp.97-108.
[17] M.J.Kim, J.T.Kim, I.J.Park, S.Y.Lee, and S.Y.Park, "Agent - Oriented Software Modeling Method with UML approach," APSEC'99, December, 1999.
[18] S. Jha, P. Chalasani, O. Shehory, and K. Sycara, "A formal treatment of distributed matchmaking," Proceedings of Agents-98, Minneapolis, Minnesota, 1998, pp.457-458.

이 승 연

1999년 서강대학교 전자계산학과(공학사). 2001년 서강대학교 대학원 컴퓨터학과(공학석사). 2001년 ~ 현재 한국전자통신연구원 컴퓨터·소프트웨어기술연구소 S/W 공학연구부 연구원. 관심분야는 소프트웨어 개발방법론, 소프트웨어 아키텍처,다중 에이전트 시스템, 컴포넌트 응용

박 수 용

1986년 서강대학교 전자계산학과 공학사. 1988 후로리다 주립대 전산학 석사. 1995년 George Mason University 정보 기술학 박사, 연구 조교수. 1996년 ~ 1998년 TRW Senior Software Engineer. 1998년 ~ 현재 서강대학교 컴퓨터학과 조교수. 관심분야는 요구공학, 에이전트 지향의 소프트웨어 공학, 소프트웨어 아키텍쳐

정 성 원

1990년 M.S. in Computer Science at Michigan State Univ. 1995년 Ph.D. in Computer Science at Michigan State Univ. 1997년 ~ 2000년 한국전산원 선임연구원. 2000년 ~ 현재 서강대학교 컴퓨터학과 조교수. 관심분야는 Mobile databases, ITS/GIS, parallel processing in database systems, spatial database, Mobile Agents