다중 에이전트 시스템 상에서 에이전트 수행 종료에 의한 문제 해결

장 명 욱† · 박 상 규† · 이 광 로† · 민 병 의†

요 약

다중 에이전트 시스템은 응용 프로그램들이 상호 협력을 통하여 문제를 해결할 수 있는 컴퓨터 수행 환경을 제공해 줌으로써 사용자가 지시해야 하는 일련의 작업을 하나의 명령으로 처리할 수 있도록 하며, 기존의 개별 응용 프로그램들을 통하여 제공할 수 없었던 보다 복잡한 명령을 간단한 형태로 지원한다. 하지만 다중 에이전트 시스템에서 에이전트들은 다른 응용 프로그램들과 공동 작업을 수행함으로써 다른 에이전트들에게 의존된다. 이러한 문제점들은 다중 에이전트 시스템이 다른 다중 에이전트 시스템들과 연계되어 수행될 때 더욱 크게 나타난다. 본 논문에서는 하나의 다중 에이전트 시스템이나 상호 연계된 다중 에이전트 시스템들 사이에서 에이전트 종료에 따르는 문제점을 기술하고, 그 해결 방법을 제시하였으며, 구현된 시스템 상에서 수행 과정에 대한 구체적인 예를 들었다.

A Solution of the Agent Termination Problem in Multi-Agent Systems

Myeong-Wuk Jang† · Sang-Kyu Park† · Gowang-Lo Lee† · Byung-Eui Min†

ABSTRACT

The multi-agent system is an environment in which applications solve tasks by cooperating with each other. In the multi-agent system, a user can solve complex problems by issuing a single command instead of a series of commands. However, agents depend on the state of other agents for they cooperate with each other. The problem worsens as multi-agent systems are connected to form a group of multi-agent systems. In this paper, we present the termination problem of an agent in a multi-agent system or a group of multi-agent systems. We also give a solution of the problem and present an example through implemened multi-agent systems.

1. 서 론

현재 많은 응용 프로그램들이 자신의 응용 분야에서 다양한 작업을 처리함으로써 사용자에게 편리한 작업 환경을 제공해 주고 있지만, 아직도 사용자가 여러 응용 프로그램들을 활용하는 작업을 처리하기 위해서는 사용자가 직접 각 응용 프로그램들을 따로 따로 수행시켜가며 일을 처리해야 하는 문제점이 있다. 그러나 사용자가 원하는 일이 현재 자신의 응용 프로그램들에 의해 처리될 수 있고, 각 응용 프로그램들이 상호 정보를 주고받으며 작업을 수행할 수 있

† 정 회 원: 한국전자통신연구소 인공지능연구실
논문접수: 1996년 9월 5일, 심사완료: 1996년 12월 2일

다면, 사용자는 직접 했던 일련의 작업을 한 번의 명령으로 처리할 수 있을 것이다.

이와 같이 응용 프로그램들의 상호 협력 방식을 통하여 사용자의 복잡한 요구사항을 처리함으로써 현재의 사용 환경보다 편리한 컴퓨터 사용 환경을 제공하기 위한 연구가 다중 에이전트 시스템에 대한 연구이다. 다중 에이전트 시스템에서 에이전트란 다른 에이전트와 상호 협력을 통하여 문제를 해결할 수 있는 능력을 갖춘 응용 프로그램을 말한다. 다중 에이전트 시스템을 사용하는 사용자는 에이전트를 통하여 기존의 독립적인 응용 프로그램들을 활용하는 것 보다 간단한 명령을 통하여 컴퓨터를 사용할 수 있고, 모든 에이전트들에 대한 구체적인 지식이 없어도 몇 개의 에이전트들만을 사용함으로써 자신의 많은 에이전트들의 기능을 활용할 수 있다. 또한 기존의 개별 응용 프로그램들로는 제공받을 수 없었던 새로운 서비스도 제공받을 수 있다[11, 12].

하지만, 에이전트는 다른 에이전트와 상호 협력을 통하여 사용자에게 서비스를 제공함으로써 다른 에이전트에 의존된다. 그래서 한 에이전트의 비정상적인 종료나 수행 중단은 다른 에이전트에게 영향을 미칠 수 있으며, 다른 에이전트의 갑작스런 수행 중단에 의해서 작업 수행에 영향을 받을 수도 있다. 하지만, 다중 에이전트 시스템은 어떠한 상황 속에서도 사용자에게 지속적이고 일관된 서비스 지원을 제공할 수 있어야 한다.

이에 본 논문에서는 직접 구현한 다중 에이전트 시스템에 대해 기술하고, 우리의 다중 에이전트 시스템이 작업을 수행함에 있어서 에이전트의 수행 종료에 의해 발생하는 문제점들에 대해 그 해결 방법을 제시하고자 한다. 이를 위하여 2장에서는 국내외 에이전트에 관한 연구 동향 및 본 연구의 연구 배경에 대해 살펴본다. 3장에서는 우리가 구현한 다중 에이전트 시스템의 구조 및 에이전트의 수행 종료에 의해 발생하는 문제들에 대해 그 해결 방법을 기술한다. 4장에서는 분산 환경에서 다중 에이전트 시스템들이 상호 연계되어 수행되기 위한 구조 및 그러한 환경에서 에이전트의 수행 종료에 의해 발생하는 문제들에 대해 그 해결 방법을 기술한다. 5장에서는 구현된 다중 에이전트 시스템 에서 에이전트 수행 종료 문제를 어떻게 대처하는지에 대해 시나리오를 기술한다. 6장에서는 본 연구에 대한 결론을 맺는다.

2. 관련 연구 및 연구 배경

다중 에이전트 시스템에 대한 연구는 크게 3 분야로 나눌 수 있다. 즉, 에이전트에 대한 이론적인 연구와 에이전트 구조에 대한 연구, 그리고 에이전트 언어에 대한 연구이다[18]. 에이전트에 대한 이론적인 연구는 에이전트에 대한 개념과 에이전트들이 가져야 할 특성, 그리고 에이전트에 대한 수학적, 논리적 표기에 대한 연구 등이 있다[9, 15, 17]. 에이전트 구조에 대한 연구는 시스템 차원에서 에이전트를 어떻게 구성할 것인가에 대한 연구이다[3, 13]. 에이전트 언어에 대한 연구는 에이전트 구현 언어와 에이전트 의미 표현 언어, 그리고 에이전트간 통신 언어로 나눌 수 있다[6, 10, 14]. 이외에 현재 가장 많은 연구가 진행되고 있는 분야로는 국내에서 가장 활발히 연구가 이루어지고 있는 분야로는 에이전트 응용 분야에 대한 연구가 있다[1, 2, 4].

에이전트 구조에 대한 연구 중 다중 에이전트 기반 구조에 대한 연구는 여러 에이전트들이 상호 협력을 통하여 문제를 해결할 수 있는 기반구조에 대한 연구이다. 현재 이에 대한 연구 결과로는 Envoy, ARCHON, OAA, EMAF 등 많은 논문이 발표되었다[3, 7, 8, 16, 19].

Envoy는 지역 망(local-area network)에서 서로 다른 시스템들 상의 응용 프로그램들이 특정 작업을 수행할 수 있도록 작업을 나누어주고, 이에 대한 수행을 감시할 수 있는 구조에 대한 연구로써 진행되었다[16]. ARCHON은 복잡한 시스템에 대하여 다른 측면으로 판단(decision making)을 하는 기존의 전문가 시스템들이 상호 협력을 할 수 있도록 구성된 다중 에이전트 기반구조이다. ARCHON 구조에서 각 에이전트는 자신이 도움을 받고자 하는 다른 에이전트들의 기능에 대한 정보를 직접 관리하게 된다[19]. OAA는 블랙보드 서버(blackboard server)라고 하는 에이전트를 중심으로 다른 에이전트들이 연계된 계층적 다중 에이전트 시스템 구조에 대한 연구로써[7], 현재 우리의 다중 에이전트 시스템 구조와 가장 유사한 구조를 갖는다[3].

위와 같은 지금까지의 다중 에이전트 기반구조에 대한 연구들은 어떻게 여러 에이전트들을 연계시키

고, 어떻게 여러 에이전트들이 상호 협력을 통하여 문제를 해결할 수 있는가에 초점을 두어 이루어 졌다. 그래서 사용자에 의해서나 시스템의 문제로 인하여 발생되는 다양한 문제들에 대해서는 그 해결 방법에 대한 연구가 아직 미진한 상황이다. 하지만 다중 에이전트 시스템에서 응용 에이전트의 추가와 삭제의 용이성은 다중 에이전트 시스템의 주요 특징으로 제시되고 있고[5, 7], 응용 에이전트의 삭제는 상호 협력을 통하여 작업을 수행하는 다중 에이전트 시스템에서 작업의 비정상적인 종료를 야기시킬 수 있다. 이에 본 논문에서는 다중 에이전트 시스템이 수행되는 상황에서 특정 에이전트가 종료됨으로써 나타나는 문제점들을 어떻게 극복할 것인가에 대해 그 해결 방법을 제시하고 있다.

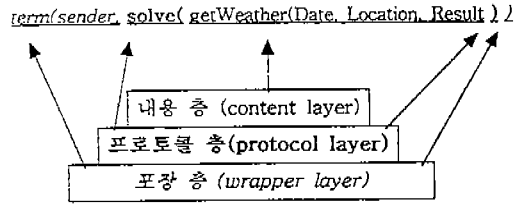
3. EMAF(Extensible Multi-Agent Framework)의 구조 및 특성

3.1 EMAF 구조

여러 에이전트들이 상호 협력하며 사용자가 요구하는 작업을 처리하기 위해서는 각 에이전트들이 다른 에이전트와 정보를 주고받을 수 있는 통신 방법이 필요하다. 이러한 통신 방법은 에이전트들이 상호 정보를 주고받는데 필요한 통신 언어(communication language)와 에이전트들이 정보를 주고받기 위해 요구되는 통신 구조(communication architecture), 그리고 에이전트들간의 통신을 위하여 에이전트들 스스로 갖추어야 할 에이전트 내부 구조(agent internal architecture)로 이루어진다. 이러한 언어 및 구조들에 대한 정의를 다중 에이전트 기반구조라고 한다. 또한 이러한 다중 에이전트 기반구조에 따라 구현된 시스템을 다중 에이전트 시스템이라고 할 수 있다.

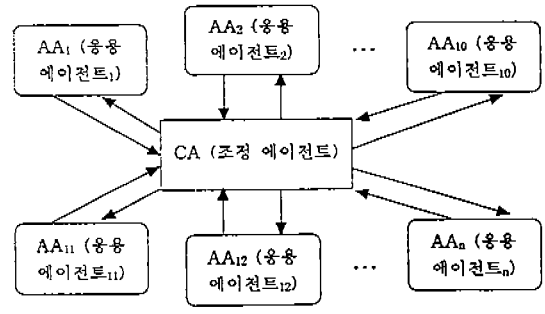
EMAF(Extensible Multi-Agent Framework)란 다중 에이전트 시스템을 만들기 위해 정의한 다중 에이전트 기반구조이다. 개발자는 EMAF의 정의에 따라 다양한 형태의 다중 에이전트 시스템을 구현할 수 있는데 이렇게 구현된 다중 에이전트 시스템을 EMAS (Extensible Multi-Agent System)라고 한다.

EMAF에서 에이전트들이 정보를 주고받기 위해 사용되는 통신언어를 ICL(Inter-agent Communication Language:에이전트간 통신언어)이라고 한다. ICL은 (그림 1)과 같이 구조적으로 선언적 형태(predicate form)를 갖으며, 의미적으로 3개의 계층적 구조를 갖는다. 즉, 내용 계층(content layer), 프로토콜 계층(protocol layer), 그리고 포장 계층(wrapper layer)으로 이루어진다. 내용 계층은 에이전트들간에 교류되는 메시지 내용을 나타낸다. 프로토콜 계층은 내용 계층에 기술된 메시지 내용에 대한 전달 의미를 포함한다. 포장 계층은 위 두 층에서 나타내지 못한 부가적인 정보를 나타내며, 현재는 메시지를 처리할 응용 에이전트를 지정하거나 현 메시지가 ICL 메시지임을 나타내는 데 사용된다.



(그림 1) ICL 구조 및 예
(Fig. 1) Architecture and example of ICL

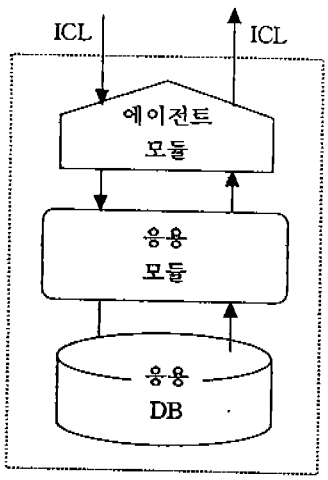
에이전트간 통신 구조는 (그림 2)와 같이 하나의 조정 에이전트(CA:Coordinating Agent)와 다수의 응용 에이전트(AA:Application Agent)로 이루어진다. EMAF에서 응용 에이전트는 하나의 응용 프로그램으로써의 고유 기능을 갖으며, 다중 에이전트 시스템에서 공동 작업을 수행한다. 조정 에이전트는 응용 에이전트들이 상호 협력을 통해 수행될 수 있도록 응용 에



(그림 2) EMAF 통신 구조
(Fig. 2) Communication architecture of EMAF

이전트들을 관리하며, 응용 에이전트들간에 정보의 전달을 담당한다.

각 에이전트들이 갖는 에이전트 내부 구조는 (그림 3)과 같이 2가지 모듈로 구성된다. 하나는 에이전트 모듈이고 다른 하나는 응용 모듈이다. 에이전트 모듈은 다른 에이전트로부터 들어온 ICL 형태의 메시지를 분석하여 해당 내용에 따라 필요한 사항을 수행시키거나, 다른 에이전트에 대하여 원하는 사항을 요구하거나 요구받은 사항에 대해 수행 결과를 전달하기 위해 내부 처리 결과를 ICL 형태로 바꾸어 주는 역할을 한다. 응용 모듈은 서비스 제공을 위하여 에이전트들이 갖는 기능을 수행하는 역할을 한다. 응용 DB는 응용 에이전트에 따라 자신의 고유 DB가 필요한 경우 이를 지원하는 역할을 한다.



(그림 3) 에이전트 내부구조
(Fig. 3) Internal architecture of agent

3.2 에이전트들간의 협력에 의한 작업 수행 방법

EMAS 상에서 응용 에이전트가 다른 에이전트의 협조를 통하여 작업을 수행하는 과정은 다음과 같다:

1. 응용 에이전트는 작업을 수행하던 도중 자신이 해결할 수 없는 기능이 있으면 이를 ICL형태로 표현하여 조정 에이전트에게 요구한다.
2. 조정 에이전트는 응용 에이전트로부터 전달된 ICL 메시지를 분석하여 요청된 기능을 지원할 수 있는 응용 에이전트를 선택한 후, 해당 응용 에이전트에게 전달받은 ICL 메시지를 전달한다.
3. 기능 수행 요청 ICL 메시지를 전달받은 응용 에이전트는 이를 분석하여 요구되어진 기능을 수행한 후 그 결과를 ICL 형태로 바꾸어 조정 에이전트에게 전달한다.
4. 조정 에이전트는 전달받은 ICL 메시지를 분석하여 전달된 ICL 메시지가 요구사항에 대한 수행 결과임을 판단하고, 기능 수행을 요청한 응용 에이전트에게 ICL 메시지를 전달한다.
5. 요구했던 사항에 대한 수행 결과를 전달받은 응용 에이전트는 전달된 수행결과를 활용하여 자신이 하고자 하는 작업의 수행을 계속한다.

위 과정에서 다른 응용 에이전트로부터 기능 수행을 요청 받은 응용 에이전트가 이를 처리하는 과정에서 다른 응용 에이전트의 도움이 필요할 경우에는 이러한 기능에 대해 다시 ICL 형태로 요구사항을 만들어 조정 에이전트에게 요청할 수 있다.

3.3 조정 에이전트와 응용 에이전트의 기능

EMAS 상에서 응용 에이전트들간의 기능 수행 요청을 다른 응용 에이전트에게 전달하고 수행 결과를 다시 이를 요청한 응용 에이전트에게 전달하기 위하여 필요한 조정 에이전트의 기능은 다음과 같다.

1) 응용 에이전트 정보 관리
2) 응용 에이전트 수행 가능 기능 관리
3) 요구사항을 지원할 수 있는 응용 에이전트 선택
4) 응용 에이전트에게 요구사항 전달
5) 전달받은 수행 결과를 이를 요청한 응용 에이전트에게 전달
6) 처리 중인 요구사항 목록 관리
7) 필요시 응용 에이전트 수행
8) 수행이 필요 없는 응용 에이전트 종료

각 응용 에이전트는 EMAS 상에서 최초 수행할 때 자신에 대한 정보 및 수행 가능 기능을 조정 에이전트에게 전달하며, 자신의 고유 기능을 수행하고, 다른 응용 에이전트의 기능을 조정 에이전트에게 요구하고 조정 에이전트로부터 요구사항에 대한 수행 결과

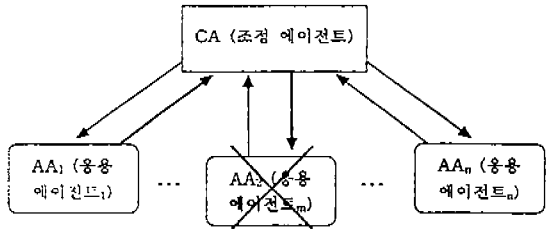
를 제공받아야 한다. 또한, 조정 에이전트를 통하여 외부로부터 요구된 사항에 대해서 이를 처리하여 그 결과를 다시 조정 에이전트에게 전달해야 한다. 이와 같은 응용 에이전트의 기능을 정리하면 다음과 같다.

1) 최초 수행할 때 자신에 대한 정보 및 수행 가능 기능을 조정 에이전트에게 등록
2) 조정 에이전트에게 요구사항 전달
3) 조정 에이전트로부터 전달받은 요구사항 수행 결과 처리
4) 조정 에이전트로부터 전달받은 요구사항 처리
5) 조정 에이전트로부터 전달받은 요구사항의 수행 결과 전달

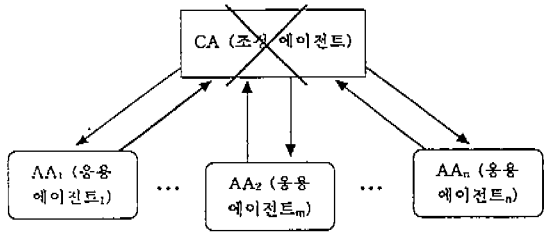
3.4 에이전트 종료에 의한 문제 및 해결 방법

EMAS 상에서 응용 에이전트는 작업을 수행함에 있어서 다른 에이전트들의 도움을 받음으로써 보다 많은 일을 처리할 수 있다. 하지만, 위와 같이 여러 응용 에이전트들이 상호간의 협력을 통하여 작업을 수행함으로써 응용 에이전트들은 함께 작업을 수행하는 다른 응용 에이전트에 의존하게 된다. 예를 들어, 다른 에이전트의 요구사항을 처리하고 있는 응용 에이전트가 갑자기 종료되었을 때, 조정 에이전트가 이미 종료된 에이전트로부터의 결과를 무작정 기다리고 있다면, 기능 수행을 요청한 응용 에이전트는 자신이 해결하고자 하는 문제를 해결할 수 없다. 이러한 에이전트의 갑작스런 종료는 사용자가 현재 어떠한 에이전트가 수행 중에 있는지를 모르고 무심코 수행 중에 있는 특정 에이전트를 종료시키는 경우나 사용자가 필요에 의해서 에이전트의 수행을 종료시키는 경우에 흔히 나타날 수 있다. 이러한 에이전트의 종료로부터 나타날 수 있는 다양한 상황들은 다음과 같은 상황들이 있다.((그림 4) 참조)

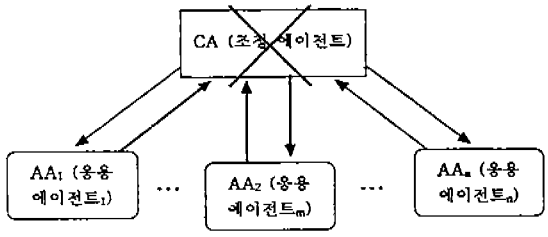
1. 조정 에이전트로부터 기능 수행을 요청 받아 작업을 수행하고 있는 응용 에이전트가 종료되는 경우
2. 조정 에이전트가 응용 에이전트로부터 기능 수행 요청을 받고 이를 다른 응용 에이전트에게 전달하기 전에 종료되는 경우
3. 조정 에이전트가 응용 에이전트의 기능 수행 요청을 다른 응용 에이전트에게 전달하고 종료되는 경우
4. 조정 에이전트에게 기능 수행을 요청을 한 응용 에이전트가 종료되는 경우



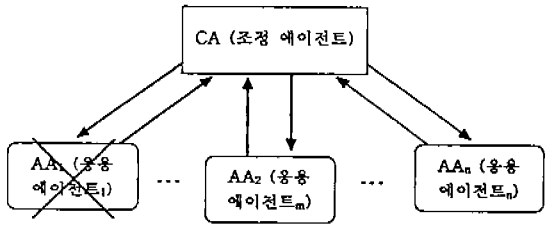
(상황 1) 작업을 수행하고 있는 응용 에이전트가 갑자기 종료되는 경우



(상황 2) 기능 수행 결과를 전달하기 전에 조정 에이전트가 종료되는 경우



(상황 3) 요구사항을 다른 응용 에이전트에게 전달하고 조정 에이전트가 종료되는 경우



(상황 4) 기능 수행을 요청한 한 응용 에이전트가 종료되는 경우

(그림 4) 에이전트 종료에 따른 문제 발생 상황들
(Fig. 4) Agent termination problems

상황 1에서 기능 수행을 요청한 응용 에이전트와 조정 에이전트는 이미 종료된 응용 에이전트로부터 기능 수행 결과가 전달되기를 한없이 기다리게 된다. 상황 2에서 응용 에이전트는 다른 응용 에이전트의 도움을 받을 수 없게 된다. 상황 3에서 기능 수행을 요청 받은 에이전트는 조정 에이전트가 종료됨에 따라 수행 결과를 전달할 수 없게 되고, 기능 수행을 요청한 응용 에이전트는 한없이 기능 수행 결과를 기다리게 된다. 상황 4에서 조정 에이전트는 기능 수행을 요청 받은 응용 에이전트로부터 결과가 전달되었을 때 이를 전달할 대상 응용 에이전트가 없어 수행 결과 처리를 할 수 없다.

위와 같은 상황들로부터 발생하는 문제들을 해결하기 위해서 기존 에이전트들의 기능이 확장되야 한다. 그러나 이러한 상황에서도 다양한 개발자에 의하여 개발된 기존의 응용 에이전트의 사용이 보장되어야 하며, 에이전트에 대해서는 비전문가인 응용 에이전트의 개발자에게 기능확장에 따른 부가적인 부담을 주지 않아야 한다. 이를 위하여 본 연구에서는 이러한 확장을 응용 에이전트에 대한 수정 없이 조정 에이전트의 확장을 통하여 처리할 수 있도록 하였다.

다음은 위 상황들을 극복하기 위한 조정 에이전트의 기능 추출을 위하여 제안된 조정 에이전트의 처리 방법이다.

상황 1의 경우에 조정 에이전트는 자신에게 연결된 응용 에이전트의 수행 상태를 지속적으로 파악하여, 현재 작업을 수행 중이던 응용 에이전트가 수행 종료되었다는 것을 인지한다. 이를 인지한 조정 에이전트는 종료된 해당 응용 에이전트를 재 수행시킬 수 있도록 시도한다. 동시에 조정 에이전트는 다른 응용 에이전트를 통하여 동일한 기능을 수행할 수 있는 에이전트가 있는지를 파악하여, 이를 지원할 수 응용 에이전트가 존재하면 해당 응용 에이전트로 하여금 이에 대한 기능을 수행하도록 요청한다. 종료되었던 응용 에이전트가 재 수행되면 조정 에이전트는 전에 요청한 기능 수행을 다시 요구한다. 이와 같이 요구된 기능에 대하여 동시에 두 에이전트에 의한 처리가 이루어져 조정 에이전트가 두 개의 기능 수행 결과들을 전달받을 수 있다. 이때에는 먼저 도착하는 수행 결과를 이를 요청한 응용 에이전트에게 전달하고, 늦게 도착하는 결과는 무시한다.

상황 2의 경우에 조정 에이전트는 전달받은 기능 수행 요청에 대하여 즉시 이를 '수행 기능 목록' 파일에 저장, 관리한다. 이 후 사용자에 의해 종료된 조정 에이전트가 재수행되면, 조정 에이전트는 전에 수행 중이었던 '수행 기능 목록' 파일을 참조하여 처리가 끝나지 않은 요구사항들에 대하여 처리 과정을 다시 시작한다.

상황 3의 경우에 기능 수행을 요청 받은 에이전트는 요청 받은 사항에 대한 처리를 마치고 그 결과를 조정 에이전트에게 전달하고자 하나 현재 조정 에이전트가 종료되어 있음으로써 수행 결과를 전달할 수 없게 된다. 이와 같은 상황에서 응용 에이전트는 자신의 수행한 결과를 버리면 된다. 사용자에 의해 조정 에이전트가 재 수행되면, 조정 에이전트는 '수행 기능 목록' 파일을 참조하여 기존에 응용 에이전트에게 요청한 사항에 대하여 해당 응용 에이전트들에게 재 수행을 요청함으로서 이 문제를 해결한다.

상황 4의 경우에 응용 에이전트의 기능 수행 요청에 대한 처리를 마친 조정 에이전트는 이를 해당 응용 에이전트에게 전달하고자 하나 해당 응용 에이전트가 종료되어 있어서 수행 결과를 전달할 수 없게 된다. 이와 같은 상황에서 조정 에이전트는 기능 수행을 요청한 응용 에이전트의 재수행을 시도하고, 전달받은 수행 결과에 대해서는 이를 보관, 관리한다. 해당 응용 에이전트가 재 수행되면 보관된 기능 수행 결과를 전달한다.

위와 같은 방법들을 통하여 앞에서 제시된 문제를 해결할 수 있다. 이를 위한 조정 에이전트의 부가적인 기능을 정리하면 다음과 같다.

1) 에이전트의 수행 상태 파악
2) 기능 수행 요구를 마친 요구사항에 대한 새로운 응용 에이전트 검색
3) 응용 에이전트에게 특정 요구사항 재수행 의뢰
4) 하나의 요구에 대한 2개 이상의 결과 처리
5) '수행 기능 목록' 파일 관리
6) 재 수행시 처리되지 않은 요구사항 처리
7) 수행 결과 보관

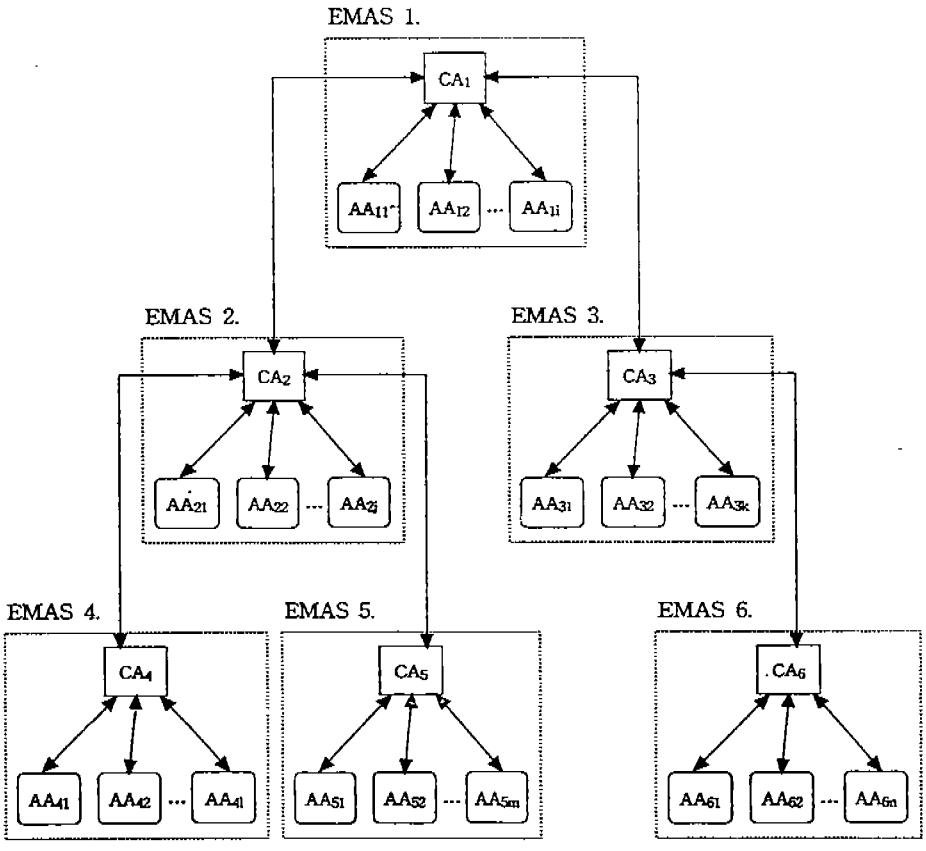
4. 분산 환경에서 EMAF의 구조 및 특성

4.1 분산 환경에서 EMAF의 구조

EMAF의 정의를 기반으로 개발자의 구현 목적에 따라 EMAS는 다양한 형태로 구현될 수 있다. 분산 환경에서 다른 개발자에 의해 개발된 EMAS들은 필요에 따라 상호 연계될 수 있다. 계속적인 EMAS들의 연계는 EMAS들의 그룹 형태로 확장된다. 이러한 연계를 통하여 사용자는 하나의 EMAS를 통해서는 얻을 수 없었던 보다 많은 서비스를 제공받을 수 있고, 어느 응용 에이전트에게 문제가 있을 경우에는 다른 EMAS에는 동일한 기능을 제공하는 응용 에이전트의 도움을 받을 수 있으며, 특정 컴퓨터 시스템 환경에 의존적인 서비스를 제공받을 수 있다. 또한, 응용 에이전트들은 하나의 EMAS내에 존재하는 응용 에이전트들의 기능들보다 많은 응용 에이전트 기능들을 활용하여 보다 다양하고 복잡한 서비스 기능들을 제공 할 수 있다.

이와 같은 EMAS들간의 연계는 조정 에이전트를 중심으로 이루어진다. 이는 EMAS들간의 연계가 응용 에이전트들간에 이루어 질 경우에 조정 에이전트에 의한 응용 에이전트의 관리가 힘들고, 에이전트 전문가가 아닌 개발자에게 응용 에이전트의 기능 확장에 따른 구현 부담이 증가하며, 현재 이미 사용 중인 응용 에이전트에 대한 수정이 요구되기 때문이다. 이러한 문제점을 극복하기 위하여 확장에 따른 기능 확장은 조정 에이전트에 의해 이루어진다.

EMAS들이 연계되어 수행되기 위한 조정 에이전트들간의 통신 구조(communication architecture)는 지역망(local area network) 환경에는 (그림 5)와 같은 계층적(hierarchy) 구조를 갖는다. 조정 에이전트는 계층적 구조상에서 위쪽으로는 한 개의 조정 에이전트와 연계되며, 아래쪽으로는 다수의 조정 에이전트와



(그림 5) EMAS들의 계층적 통신 구조
(Fig. 5) Communication architecture of EMAS's

연계된다. 이때, 계층적 구조상에서 조정 에이전트 위쪽에 위치한 조정 에이전트를 상위 조정 에이전트라고 하고, 아래쪽에 위치한 조정 에이전트를 하위 조정 에이전트라 한다. 그래서, 임의의 조정 에이전트1이 조정 에이전트2의 상위 조정 에이전트이면, 그 조정 에이전트2는 조정 에이전트1의 하위 조정 에이전트가 된다.

4.2 조정 에이전트 작업 수행 방법

EMAS들이 계층적 구조로 연계된 상황에서 조정 에이전트에 의한 작업 수행 과정을 살펴보면 다음과 같다.

1. 응용 에이전트는 수행 과정에서 요구되는 자신이 해결할 수 없는 기능 수행에 대해 이를 조정 에이전트에게 전달한다. 이때 전달되는 메시지의 형태는 ICL 형식을 따른다.
2. 조정 에이전트는 응용 에이전트로부터 전달받은 기능 수행 요청에 대해 자신의 응용 에이전트들 중 하나에 의하여 이를 지원할 수 있는지 조사한다.
3. 만약 자신의 응용 에이전트에 의해 요구된 기능을 지원할 수 있는 경우에 조정 에이전트는 해당 응용 에이전트에게 기능 수행을 요청하여 처리한 후 수행 결과를 이를 요청한 조정 에이전트에게 전달한다.
4. 만약 자신의 응용 에이전트에 의해 요구된 기능을 지원할 수 없다면, 조정 에이전트는 이를 자신의 하위 조정 에이전트 중 하나에게 전달한다.

5-a. 기능 수행 요청을 전달받은 하위 조정 에이전트는 자신의 응용 에이전트들 중에서 이를 지원할 수 있는 응용 에이전트를 찾는다.

5-b. 만약 하위 조정 에이전트의 응용 에이전트를 통하여 요구된 기능 수행을 지원할 수 있다면, 해당 응용 에이전트에 의하여 요구된 사항을 처리한 후 수행 결과를 이를 요청한 상위 조정 에이전트에게 전달한다.

5-c. 만약 하위 조정 에이전트의 어느 응용 에이전트도 기능 수행 지원을 할 수 없는 경우에는 동일한 방식으로 자신의 하위 조정 에이전트 중 하나에게 이를 의뢰한다. 이때, 하위 조정 에이전트에게 그의 하위 조정 에이전트가 존재하지 않는 경우에는 기능 수행을 의뢰한 조정 에이전트에게 이를 처리할 수 없음을 알린다.

5-d. 하위 조정 에이전트의 모든 하위 조정 에이전트가 요구된 기능 수행을 지원할 수 없다면, 기능 수행을 의뢰한 조정 에이전트에게 이를 처리할 수 없음을 알린다.

5-e. 하위 조정 에이전트의 하위 조정 에이전트 수행 방식은 5-a에서 5-d의 과정을 반복한다.

6. 하위 조정 에이전트로부터 요구사항을 처리할 수 없다는 메시지를 전달받은 조정 에이전트는 자신의 다른 하위 조정 에이전트에게 이를 의뢰한다. 이에 대한 수행 과정은 5-a에서 5-e 과정과 동일하다.
7. 만약 자신의 다른 하위 조정 에이전트가 없거나 모든 하위 조정 에이전트들로부터 요구사항을 처리할 수 없다는 메시지가 전달되면, 조정 에이전트는 자신의 상위 조정 에이전트에게 기능 수행을 의뢰한다.

8-a. 기능 수행 요청을 전달받은 상위 조정 에이전트는 먼저 자신의 응용 에이전트들 중에서 이를 처리할 수 있는 응용 에이전트를 찾는다.

8-b. 만약 상위 조정 에이전트의 특정 응용 에이전트를 통하여 요구된 기능 수행 요청을 지원할 수 있다면, 해당 응용 에이전트에 의하여 요구된 사항을 처리한 후 그 결과를 이를 요청한 하위 조정 에이전트에게 전달한다.

8-c. 만약 상위 조정 에이전트도 자신의 응용 에이전트들로부터 이를 지원할 수 있는 응용 에이전트를 찾지 못한다면, 상위 조정 에이전트는 자신의 다른 하위 조정 에이전트들 중에서 이를 처리 할 수 있는 조정 에이전트가 있는지 순서적으로 요구사항 처리를 의뢰한다. 이에 대한 수행 과정은 5-a에서 5-e과정과 동일하다.

8-d. 상위 조정 에이전트 역시 자신의 하위 조정 에이전트들 중에서 요구된 기능을 처리할 수 있는 조정 에이전트를 찾지 못할 경우에는 다시 자신의 상위 조정 에이전트에게 이에 대한 처리를 의뢰한다.

8-e. 상위 조정 에이전트의 상위 조정 에이전트 수행 방식은 8-a에서 8-d의 과정을 반복한다.

9. 위와 같은 반복과정을 통하여 요구된 기능 수행 요청을 지원할 수 있는 응용 에이전트를 찾게 되면, 해당 응용 에이전트로부터 수행 결과를 구한 후 이를 처음 요구한 조정 에이전트에게 전달한다.
10. 기능 수행 요청을 처음 요구한 조정 에이전트는 수행 결과가 전달되면 이를 요청한 응용 에이전트에게 전달한다.

위와 같이 하나의 요구사항에 대해 하위 조정 에이전트와 상위 조정 에이전트의 도움을 받아 요구사항을 처리할 수 있는 응용 에이전트를 찾는 과정은 처음 기능 수행을 요청한 조정 에이전트로부터 밑으로 얼마나 요구사항에 대한 의뢰를 시도할 것인지와 위로 얼마나 요구사항에 대한 의뢰를 시도할 것인지에 대한 정보와 함께 기능 수행 요청에 대한 ICL 메시지가 이동된다. 이는 요구사항의 전달이 아래나 위로 무한정 전달됨으로써 컴퓨터 네트워크이 에이전트간 요구사항 전달로 폭주하는 것을 막기 위해서이다.

4.3 조정 에이전트의 추가 기능

조정 에이전트가 스스로 해결할 수 없는 문제를 다른 조정 에이전트들의 도움을 받아 처리하기 위해서 필요한 조정 에이전트의 부가적인 기능은 다음과 같다.

1) 상위 조정 에이전트 관리
2) 하위 조정 에이전트들 관리
3) 하위 조정 에이전트에게 요구사항 의뢰 기능
4) 상위 조정 에이전트에게 요구사항 의뢰 기능
5) 요구사항을 처음 요청한 조정 에이전트로부터의 요구사항 이동 거리 관리 기능

4.4 에이전트 종료에 의한 문제 및 해결 방법

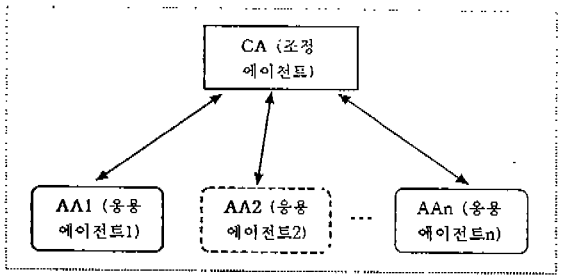
하나의 EMAS가 다른 EMAS들과 연계되어 확장되는 경우에 이를 정상적으로 운영하기 위한 방법은 하나의 EMAS에서보다 더 복잡해진다. EMAS들끼리 확장된 상황에서 이제 에이전트들은 같은 시스템에 존재하는 다른 에이전트들에게만 의존되는 것이 아니라 연계되어 수행되는 다른 시스템의 에이전트들에게도 의존된다. 이러한 의존도는 에이전트가 하나의 시스템 상에서 수행될 때와 비교하여 보다 더 많은 문제 발생 가능성을 갖는다. 예를 들어, 어느 EMAS가 다른 EMAS의 요청을 받아 작업을 수행하고 있는 상황에서 사용자가 자신의 시스템을 종료시키는 경우나 다른 EMAS의 요청을 받아 작업을 수행하고 있는 응용 에이전트를 사용자가 종료시키고자 하는 경우 등이 있다. 이러한 다양한 경우들에 대해서도 EMAF는 남아 있는 EMAS들의 응용 에이전트들을 통하여 지속적으로 작업이 수행할 수 있도록 해야 한다. 이를 위하여 이러한 상황들에 대하여 이를 해결할 수 있도록 EMAF를 확장하는 것이 필요하다.

EMAF를 확장하기 위하여 다중 에이전트 시스템들이 연계된 환경에서 특정 에이전트가 수행 종료됨으로써 발생할 수 있는 상황들에 대해 살펴보면 다음과 같다.((그림 6) 참조)

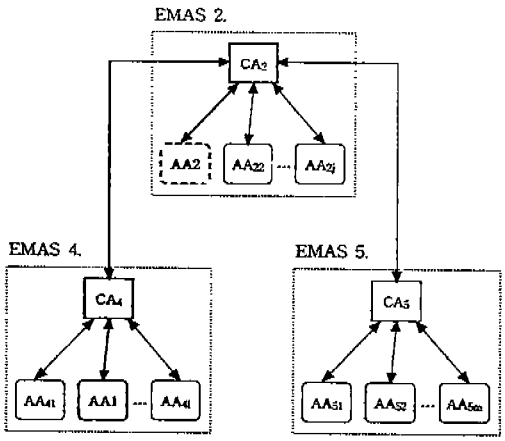
1. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 다른 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트2가 수행을 중단한 경우
2. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 다른 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트1이 수행을 중단한 경우
3. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 다른 응용 에이전트2의 도움을 받고 있는 도중 조정 에이전트가 수행을 중단한 경우
4. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 상위 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트2가 수행을 중단한 경우
5. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 상위 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트1이 수행을 중단한 경우
6. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 상위 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트2를 제어하는 조정 에이전트가 수행을 중단한 경우
7. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 상위 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트1을 제어하는 조정 에이

전트가 수행을 중단한 경우

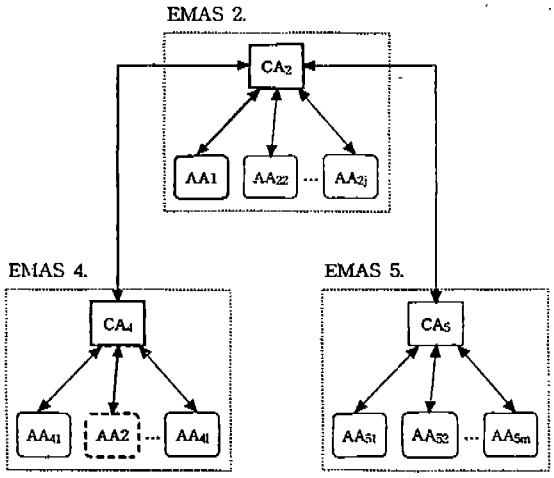
8. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 하위 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트2가 수행을 중단한 경우
9. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 하위 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트1이 수행을 중단한 경우
10. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 하위 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트2를 제어하는 조정 에이전트가 수행을 중단한 경우
11. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 하위 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트1을 제어하는 조정 에이전트가 수행을 중단한 경우
12. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 상위의 상위에 있는 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트2가 수행을 중단한 경우
13. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 상위의 상위 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트1이 수행을 중단한 경우
14. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 상위의 상위 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트2를 제어하는 조정 에이전트가 수행을 중단한 경우
15. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 상위의 상위 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트1을 제어하는 조정 에이전트가 수행을 중단한 경우
16. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 상위의 상위 다중 에이전



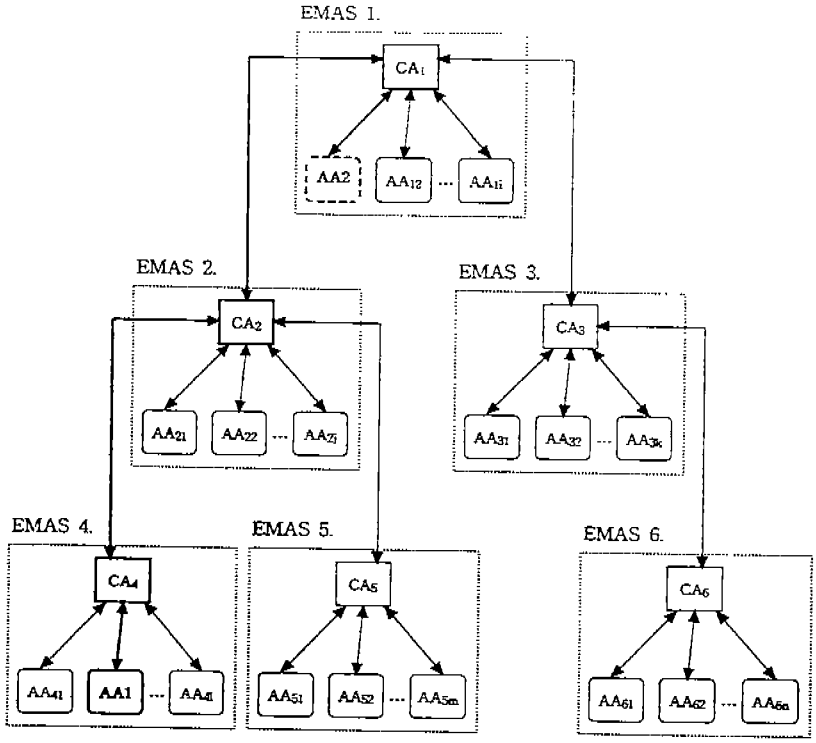
(a)상황 1~3의 경우



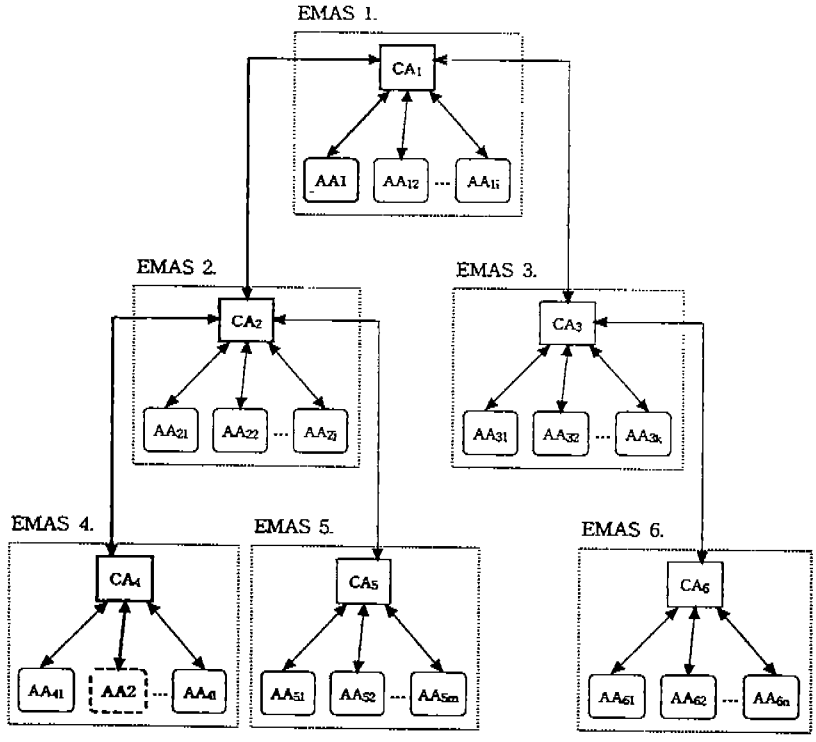
(b)상황 4~7의 경우

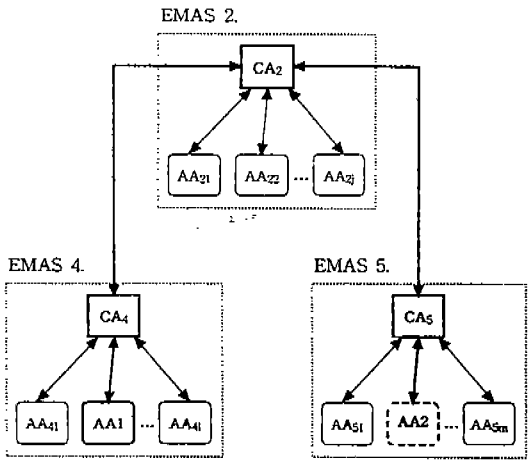


(c)상황 8~11의 경우


EMAS 1.
CA1
AA2
AA12
AA1i
EMAS 2.
CA2
AA21
AA22
AA2j
EMAS 3.
CA3
AA31
AA32
AA3k
EMAS 4.
CA4
AA41
AA1
AA4l
EMAS 5.
CA5
AA51
AA52
AA5m
EMAS 6.
CA6
AA61
AA62
AA6n

(d)상황 12~16의 경우


EMAS 1.
CA1
AA1
AA12
AA1i
EMAS 2.
CA2
AA21
AA22
AA2j
EMAS 3.
CA3
AA31
AA32
AA3k
EMAS 4.
CA4
AA41
AA2
AA4l
EMAS 5.
CA5
AA51
AA52
AA5m
EMAS 6.
CA6
AA61
AA62
AA6n



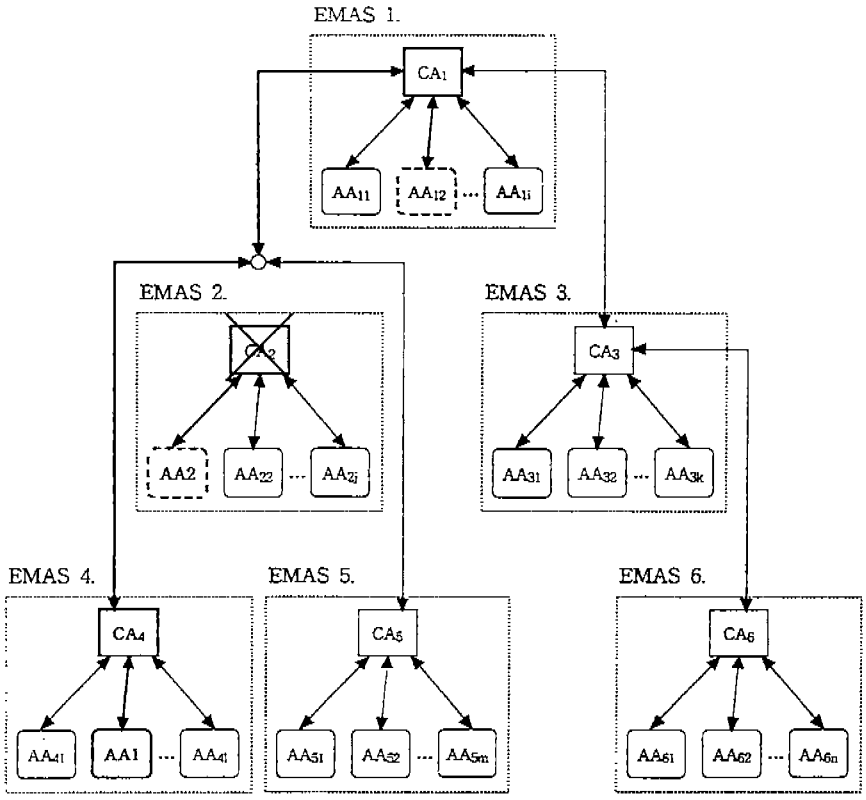
(f)상황 22~26의 경우

(그림 6) EMAS들 상에서의 통신 경로
(Fig. 6) Communication path on EMAS's

트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 양 쪽 조정 에이전트들 사이에 있는 조정 에이전트가 수행을 중단한 경우

17. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 하위의 하위에 있는 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트2가 수행을 중단한 경우
18. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 하위의 하위 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트1이 수행을 중단한 경우
19. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 하위의 하위 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트2를 제어하는 조정 에이전트가 수행을 중단한 경우
20. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 하위의 하위 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트1을 제어하는 조정 에이전트가 수행을 중단한 경우
21. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 하위의 하위 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 양 쪽 조정 에이전트들 사이에 있는 조정 에이전트가 수행을 중단한 경우
22. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 상위의 상위에 있는 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트2가 수행을 중단한 경우
23. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 상위의 다른 하위 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트1이 수행을 중단한 경우
24. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 상위의 다른 하위 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트2를 제어하는 조정 에이전트가 수행을 중단한 경우
25. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 상위의 다른 하위 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 응용 에이전트1을 제어하는 조정 에이전트가 수행을 중단한 경우
26. 하나의 다중 에이전트 시스템 내에 있는 응용 에이전트1이 자신의 상위의 다른 하위 다중 에이전트 시스템 내에 있는 응용 에이전트2의 도움을 받고 있는 도중 두 조정 에이전트의 상위 조정 에이전트가 수행을 중단한 경우

위 상황들 중 1, 4, 8, 12, 17, 22는 모두 비슷한 경우로써 응용 에이전트2의 조정 에이전트는 일정 시간 후 응용 에이전트2의 재수행을 통하여 작업의 계속적인 진행을 시도한다. 또한 동일한 기능을 갖는 다른 에이전트를 통하여 기능 수행 요구에 대한 처리를 시도한다. 하지만, 자신의 EMAS 내에 이를 지원할 다른 응용 에이전트가 없을 경우 EMAS는 먼저 연계된 하위 조정 에이전트에게 순차적으로 이에 대한 수행을 의뢰한다. 만약 EMAS에 연계된 하위 조정 에이전트가 없거나 존재하는 하위 조정 에이전트들로부터



(그림 7) 상황 6으로부터 형성된 임시적인 계층 구조
(Fig. 7) Temporal hierachical structure from state 6

이에 대한 지원이 불가능하다는 메시지가 전달되면, EMAS는 자신의 상위 조정 에이전트에게 이에 대한 수행을 의뢰한다. 이러한 과정을 통하여 수행 결과가 2개 이상 전달되면 응용 에이전트2의 조정 에이전트는 첫 번째 결과만을 활용하고 2번째 이후의 결과들은 무시한다.

위 상황들 중 2, 5, 9, 13, 18, 23들은 또한 비슷한 경우들로써 응용 에이전트1의 조정 에이전트는 요구된 기능 수행 결과를 보관하며, 응용 에이전트1이 재 수행되기를 반복적으로 시도하다가 응용 에이전트1이 재 수행되면 수행 결과를 전달한다.

위 상황들 중 3, 6, 10, 14, 19, 24들과 같은 경우에 조정 에이전트는 자신이 종료됨을 상위 및 하위 조정 에이전트에게 알린다. 종료될 조정 에이전트의 상위 조정 에이전트는 종료될 조정 에이전트의 하위 조정 에이전트들과 임시적인 계층 구조를 형성함으로써 종료될 조정 에이전트의 하위 조정 에이전트들을 자신의 하위 조정 에이전트와 같이 활용하게 된다. 하지만, 상황 3은 응용 에이전트2의 조정 에이전트가 다시 수행될 때까지 응용 에이전트1은 자신이 요청한 요구사항에 대해 그 수행 결과를 전달받을 수 없다. (그림 7)은 상황 6으로부터 형성된 임시적 계층 구조를 보여준다. 응용 에이전트1은 조정 에이전트가 재 수행된 후 이에 대한 재처리 과정을 거쳐 수행 결과를 전달받게 된다. 상황 6, 14의 경우는 응용 에이전트2의 조정 에이전트의 상위 조정 에이전트가 응용 에이전트2의 조정 에이전트가 책임져야 할 기능에 대해서 책임을 진다. 이때 만약 응용 에이전트2의 조정 에이전트의 상위 조정 에이전트가 존재하지 않으면 응용 에이전트1의 조정 에이전트는 응용 에이전트2의 조정 에이전트가 재 수행될 때까지 기다려야 한다. 상황 10, 19의 경우에는 응용 에이전트2의 조정 에이전트의 상위 조정 에이전트(상황 10의 경우 이 조정 에이전트는 응용 에이전트1의 조정 에이전트가

됨)가 다시 이에 대한 책임을 지게 된다. 그래서 아직 요구사항을 의뢰하지 않았던 다른 하위 조정 에이전트가 있는지를 검색하여 이에 대한 수행을 요청하고, 만약 존재하지 않으면, 조정 에이전트간 요구사항 의뢰의 기본 순서에 따라 자신의 상위 조정 에이전트(상황 19의 경우 이 조정 에이전트는 응용 에이전트1의 조정 에이전트가 됨)에게 이에 대한 수행을 의뢰한다. 상황 24의 경우에는 응용 에이전트2의 조정 에이전트의 상위 조정 에이전트가 다시 이에 대한 책임을 진다.

상황 7, 11, 15, 20, 25와 같은 경우에 응용 에이전트1의 조정 에이전트는 자신이 종료됨을 자신의 상위 에이전트와 하위 에이전트들에게 알리고 종료됨으로써, 상위 에이전트들과 하위 에이전트들이 임시적인 계층 구조를 형성할 수 있도록 한다. 하지만, 이와 같은 경우에 어떠한 방법으로도 응용 에이전트1에게 수행 결과가 전달 될 수 없는데 이에 대해서는 특정 조정 에이전트가 책임을 진다. 상황 7, 15, 25의 경우에는 응용 에이전트1의 상위 조정 에이전트가 수행 결과 보관 및 추후 전달에 대한 책임을 진다. 상황 11의 경우에는 응용 에이전트2의 조정 에이전트가 책임을 진다. 상황 20의 경우에는 응용 에이전트1의 하위 조정 에이전트이며 응용 에이전트2의 조정 에이전트의 상위 조정 에이전트인 조정 에이전트가 이에 대한 책임을 진다.

상황 16, 21과 같은 경우에 종료되는 조정 에이전트는 자신의 상위 조정 에이전트와 하위 조정 에이전트에게 자신이 종료됨을 알린다. 종료되는 조정 에이전트의 상위 조정 에이전트와 하위 조정 에이전트들 간에는 임시적인 계층 구조가 형성된다. 이때 두 조정 에이전트는 응용 에이전트1의 조정 에이전트와 응용 에이전트2의 조정 에이전트로써 기능 수행 결과를 전달하는데 있어서 아무런 문제가 없다. 상황 26과 같은 경우에 종료되는 조정 에이전트의 상위 조정 에이전트가 존재하면, 상위 조정 에이전트를 중심으로 응용 에이전트1의 조정 에이전트와 응용 에이전트2의 조정 에이전트간에 임시적인 계층 구조가 형성되며, 기능 수행 상에는 아무런 문제가 없다. 이때 이 두 조정 에이전트간의 중계 역할은 종료되는 조정 에이전트의 상위 조정 에이전트가 된다. 하지만 종료되는 조정 에이전트의 상위 조정 에이전트가 존재하지 않는 경우에 두 조정 에이전트들은 분리되고, 종료된 조정 에이전트가 재수행되기까지 수행 결과는 전달할 수 있는 방법이 없다. 이때 응용 에이전트1의 조정 에이전트는 현재 다른 방법을 통하여 요구된 기능을 처리할 수 없으므로 응용 에이전트1에게 서비스 수행이 불가능하다는 메시지를 전달한다.

지금까지 분산환경에서 에이전트가 종료됨으로써 나타날 수 있는 상황에 대한 문제들과 그에 대한 해결책을 살펴보았다. 분명 이는 수많은 EMAS들이 수많은 층으로써 상황 연계된 환경 상에서 나타날 수 있는 문제들과 대응 방법은 아니다. 하지만, 그러한 상황도 위 상황의 확장으로 고려될 수 있으며, 밑에서 제시될 조정 에이전트의 확장 기능을 통하여 동일하게 처리될 수 있을 것임을 알 수 있을 것이다.

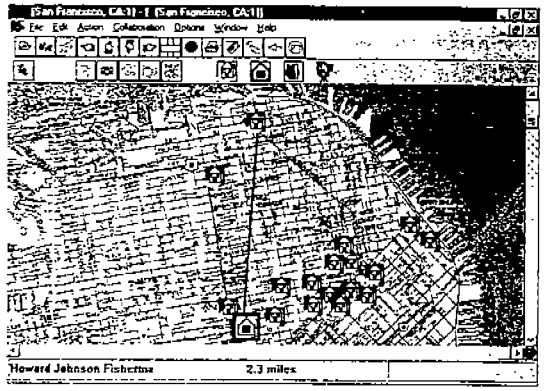
위의 문제 해결 방법으로부터 유추할 수 있는 조정 에이전트 확장 기능은 다음과 같다.

1) 조정 에이전트가 종료될 때 이에 대한 정보를 자신의 하위 조정 에이전트들과 상위 조정 에이전트에게 전달
2) 조정 에이전트가 종료될 때 다른 조정 에이전트로부터 전달받은 요구사항에 대한 처리를 상위 조정 에이전트에게 의뢰
3) 하위 조정 에이전트로부터 자신이 곧 종료될 것이라는 정보가 전달되면 종료될 하위 조정 에이전트의 하위 조정 에이전트들과 임시적인 계층 구조를 형성
4) 종료되었던 조정 에이전트가 다시 수행되면 자신의 상위 조정 에이전트와의 대화를 통하여 원래의 계층 구조를 회복

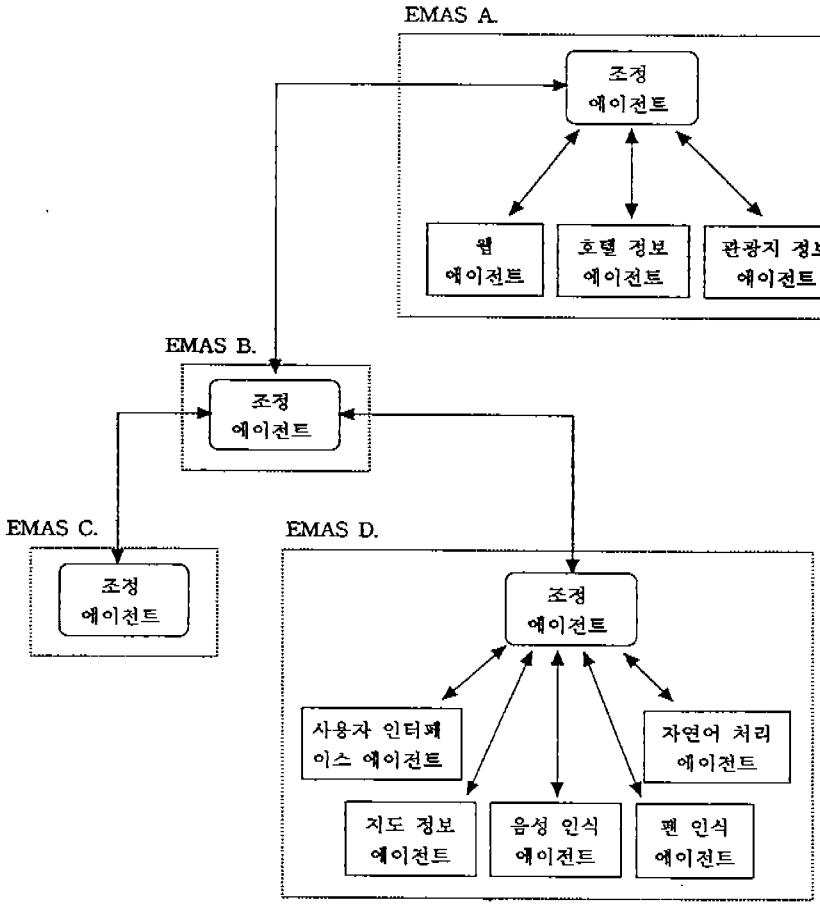
5. 구 현

에이전트 수행 종료에 의한 문제 해결 방법에 대한 점검을 위하여 구현된 다중 에이전트 시스템들은 SunOS 4.1.3이나 Windows NT 3, 51, 그리고 Windows 95 상에서 구현되었다. 이 중 다중 에이전트 시스템들의 조정 에이전트는 Quintus Prolog로 구현되었으며, 응용 에이전트들은 C나 Delphi로 구현되었다. 작업을 수행하는 에이전트들은 TCP/IP를 기반으로 하는 socket을 통하여 통신을 하도록 하였다.

일반 사용자가 다중 에이전트 시스템을 사용하는 방법은 개별 응용 에이전트들을 통하여 이용하는 방법이 있다. 하지만, 이 방식은 사용자가 다양한 종류의 응용 에이전트를 모두 다룰 수 있어야 한다는 문제점이 있다. 또한, 사용자가 다른 시스템에 존재하는 응용 에이전트를 사용하기 어렵다는 문제점이 있다. 그래서, 일반 사용자들이 다중 에이전트 시스템을 보다 편리하게 사용할 수 있도록 지원하는 특수 목적의 응용 에이전트를 개발하였는데, 이를 사용자 인터페이스 에이전트라고 한다. 사용자는 사용자 인터페이스 에이전트를 통하여 다른 응용 에이전트들의 기능을 보다 쉽게 사용할 수 있다. (그림 7)은 현재 구현된 사용자 인터페이스 에이전트의 한 예를 보여주고 있다. 본 사용자 인터페이스 에이전트는 지도를 기반으로 여행 서비스를 제공하며, 다른 응용 에이전트들의 협력을 통하여 다양한 서비스를 제공한다.



(그림 7) 지도 기반 사용자 인터페이스 에이전트
(Fig. 7) Map based user interface agent



(그림 8) 시나리오를 위한 EMAS들의 계층적 통신 구조
(Fig. 8) Communication architecture of EMAS's for a scenario

지도 기반 사용자 인터페이스 에이전트를 통하여 제공되는 정보로는 주요 호텔, 식당, 관광지에 대한 문자 및 영상 정보와 두 지역간의 거리 정보, 그리고 주요 위치에 대한 웹 정보 등이 있다. 이러한 서비스를 제공하기 위해서는 다양한 위치에 존재하는 다양한 다중 에이전트 시스템들이 상호 협력을 통하여 작업을 수행해야 한다.

이와 같이 여러 다중 에이전트 시스템이 연계되어 수행되는 상황에서 특정 다중 에이전트가 수행 종료됨으로써 서비스를 요청한 다중 에이전트 시스템에게 미치는 영향을 알아보기 위해 시험한 내용 중 하나의 시나리오를 기술하면 다음과 같다.

본 시나리오에서 사용된 다중 에이전트 시스템들은 (그림 8)과 같이 모두 4개 시스템들이 있다. 사용자가 사용을 시작하는 D 시스템은 지도 기반 사용자 인터페이스 에이전트와 음성 인식 에이전트, 펜 인식 에이전트, 자연어 처리 에이전트, 지도 정보 에이전트, 그리고 조정 에이전트로 이루어져 있으며, B, C 시스템은 조정 에이전트만 존재하고, A 시스템은 웹 에이전트, 호텔 정보 에이전트, 관광지 정보 에이전트 등으로 이루어져 있다.

1. 지도 기반 사용자 인터페이스 에이전트를 사용하는 사용자는 지도상에 펜으로 '호텔들을 보여줘'라고 입력한다.
2. 사용자 인터페이스 에이전트는 입력된 펜 입력 정보를 조정 에이전트에게 전달한다.
3. 조정 에이전트는 사용자의 펜 입력 정보를 펜 인식 에이전트에게 전달한다.
4. 펜 인식 에이전트는 펜 입력 정보를 분석하여 '호텔들을 보여줘'라는 문자 정보로 변환한다. 펜 인식 에이전트는 인식된 문자 정보를 조정 에이전트에게 전달한다.
5. 조정 에이전트는 펜 인식한 문자 정보를 자연어 처리 에이전트에게 전달한다.
6. 자연어 처리 에이전트는 문자 정보를 처리하여 show_hotels라는 ICL의 내용층 요구사항 표현 형태로 변환한다. 자연어 처리 에이전트는 변환된 요구사항을 조정 에이전트에게 전달한다.
7. 조정 에이전트는 요구사항을 지원하기 위한 응용 에이전트를 찾아 보지만, 자신의 응용 에이전트들로부터 찾지 못한다.
8. 조정 에이전트는 자신의 하위 조정 에이전트를 찾아 요구사항을 해결하려 하지만, 하위 조정 에이전트를 찾지 못한다.
9. 시스템 D의 조정 에이전트는 시스템 B에 있는 상위 조정 에이전트에게 요구사항을 해결해 줄 것을 요청한다.
10. 시스템 B의 조정 에이전트는 요구사항을 지원하기 위한 응용 에이전트를 찾아 보지만, 응용 에이전트를 찾지 못한다.
11. 시스템 B의 조정 에이전트는 시스템 C의 하위 조정 에이전트에게 요구사항을 해결해 줄 것을 요청한다.
12. 시스템 C의 조정 에이전트는 요구사항을 지원하기 위한 응용 에이전트를 찾아 보지만, 응용 에이전트를 찾지 못한다.
13. 시스템 C의 조정 에이전트는 자신의 하위 조정 에이전트를 찾아 요구사항을 해결하려 하지만, 하위 조정 에이전트를 찾지 못한다.
14. 시스템 C의 조정 에이전트는 시스템 B의 조정 에이전트에게 요구사항을 처리할 수 없음을 알린다.
15. 시스템 B의 조정 에이전트는 자신의 다른 하위 조정 에이전트를 통하여 요구사항을 해결하려 하지만, 다른 하위 조정 에이전트를 찾지 못한다.
16. 시스템 B의 조정 에이전트는 시스템 A에 있는 상위 조정 에이전트에게 요구사항을 해결해 줄 것을 요청한다.
17. 시스템 A의 조정 에이전트는 호텔 정보 에이전트에게 요구사항을 전달한다.
18. 호텔 정보 에이전트는 호텔들의 위치 정보 목록을 조정 에이전트에게 전달한다.
19. 시스템 A의 조정 에이전트는 호텔들의 위치 정보 목록을 시스템 B의 조정 에이전트에게 전달한다.
20. 시스템 B의 조정 에이전트는 호텔들의 위치 정보 목록을 시스템 C의 조정 에이전트에게 전달한다.
21. 시스템 C의 조정 에이전트는 호텔들의 위치 정보 목록을 사용자 인터페이스 에이전트에게 전달한다.

22. 지도 기반 사용자 인터페이스 에이전트는 전달된 호텔들의 위치 정보 목록을 참조하여 지도 윈도우 상에 호텔들을 표시한다.

이와 같이 사용자는 필요한 응용 에이전트가 자신의 시스템에 존재하지 않더라도 연결되어 있는 다른 다중 에이전트 시스템의 도움을 받을 수 있다. 여기서 에이전트 수행 종료에 대한 문제 해결 방법을 점검하기 위하여 사용자의 요구사항이 시스템 A의 조정 에이전트에게까지 전달되었을 때, 시스템 B를 수행 중단 시켜 보았다. 이때의 처리 과정은 위 과정에서 17번 단계부터 다음과 같이 변화되어 수행되었다.

17. 시스템 B의 조정 에이전트는 종료되기 전에 시스템 A의 조정 에이전트에게 자신이 곧 종료될 것이라는 정보와 하위 조정 에이전트들에 대한 정보를 알린다.
18. 시스템 A의 조정 에이전트는 시스템 C와 시스템 D에 있는 조정 에이전트들에게 상위 조정 에이전트가 바뀌었음을 알린다.
19. 시스템 C에 있는 조정 에이전트는 상위 조정 에이전트에 대한 정보를 바꾼다.
20. 시스템 D에 있는 조정 에이전트는 상위 조정 에이전트에 대한 정보를 바꾼다.
21. 시스템 A의 조정 에이전트는 호텔 정보 에이전트에게 요구사항을 전달한다.
22. 호텔 정보 에이전트는 호텔들의 위치 정보 목록을 조정 에이전트에게 전달한다.
23. 시스템 A의 조정 에이전트는 호텔들의 위치 정보 목록을 시스템 C의 조정 에이전트에게 전달한다.
24. 시스템 C의 조정 에이전트는 호텔들의 위치 정보 목록을 사용자 인터페이스 에이전트에게 전달한다.
25. 지도 기반 사용자 인터페이스 에이전트는 전달된 호텔들의 위치 정보 목록을 참조하여 지도 윈도우 상에 호텔들을 표시한다.

6. 결 론

지금까지 다중 에이전트 시스템을 구현하기 위해 필요한 다중 에이전트 기반구조를 제시하였고, 다중 에이전트 시스템 수행 과정에서 에이전트를 종료시킴으로써 파생되는 문제들을 기술하였으며, 그 해결 방법을 제시하였다. 그리고 연계된 다중 에이전트 시스템들의 수행 과정에서 에이전트 종료에 의해 파생되는 문제점들을 기술하였고, 그 해결 방법들을 기술하였다. 마지막으로 구현된 다중 에이전트 시스템에서 에이전트 수행 종료 문제를 해결하는 예를 시나리오를 통해 기술하였다.

본 논문에서 제시된 분산 환경에서 에이전트 종료로부터 나타날 수 있는 문제의 해결 방법은 응용 에이전트에 대한 수정을 요구하지 않는다. 이는 에이전트 비전문가인 응용 에이전트 개발자들에게 에이전트에 대한 부가적인 부담을 주지 않으며, 이미 사용하고 있는 응용 에이전트에 대한 수정이 요구하지 않음으로써 다중 에이전트 시스템의 기능 확장으로 인한 사용자의 부담을 최소화시킬 수 있다는 특징을 갖는다.

하지만, 앞에서 다루어진 내용은 분산 환경에서 다중 에이전트 시스템들이 연계되어 수행될 때 각 응용 에이전트가 사용자의 종료 지시에 따라 정상적으로 종료되는 것을 기반으로 한다. 그래서 시스템의 갑작스런 종료나 네트워크 상의 문제 발생 등과 같은 비정상적인 종료에 대한 문제에 대해서는 앞에서의 해결 방법만으로는 완전하지 않다. 또한 앞에서 제시된 해결 방법은 네트워크 상에 연결된 여러 개의 다중 에이전트 시스템들이 거의 수행되고 있지 않은 상황에서 다중 에이전트 시스템이 수행을 시작할 때, 다중 에이전트 시스템의 조정 에이전트가 자신의 상위 조정 에이전트와 하위 조정 에이전트들을 찾을 수 없어 시스템들의 연계가 여러 그룹 형태로 나타날 수 있다는 문제점을 안고 있다. 이러한 문제점을 최소화시키기 위해서는 제시된 다중 에이전트 시스템들을 지속적으로 동작하고 있는 사무실이나 연구실 환경에서 사용하는 것이 필요하다. 하지만, 이러한 환경은 요즘과 같이 사무실이나 가정으로 PC가 보급되고 있는 상황에서 일반적이라고 보기 어렵다. 따라서 이와 같은 문제점들을 극복하기 위한 연구가 앞으로 계속적으로 이어져야 할 것이다.

참 고 문 헌

[1] 기세훈, 전혜정, 강윤선, 이희연, "자연어 인터페

이스를 가진 일정관리 에이전트," HCI '96 학술대회발표논문집, 제 5권, 제 1호, pp. 116-127, 1996년.

[2] 김병학, 이광형, 조충호, "정보 검색을 위한 인텔리전트 웹 에이전트," 정보과학회지, 제 14권, 제 4호, pp. 12-21, 1996년.

[3] 백순철, 최중민, 장명욱, 박상규, 임영환, "이형 분산 환경에서 에이전트들간의 이형성을 극복하기 위한 멀티에이전트 기반구조," 정보과학회논문지(C), 제 2권, 제 1호, pp. 24-37, 1996년.

[4] 장명욱, 박상규, 신동욱, "통신 서비스 사용을 위한 에이전트 시스템에 대한 연구," '94 정보과학회 가을 학술발표논문집, 제 21권, 제 2호, pp. 715-718, 1994년.

[5] 백순철, 최중민, 장명욱, 박상규, 민병의, 임영환, "플러그 앤드 플레이(Plug and Play) 개념을 이용한 이형 응용 프로그램의 통합 기법," 정보처리논문지, 제 2권, 제 6호, pp. 98-110, 1995년.

[6] Mihai Barbuceanu and Mark S. Fox, "COOL: A Language for Describing Coordination in Multi Agent Systems," Proceedings of the First International Conference on Multi-Agent Systems, pp. 17-24, San Francisco, June 1995.

[7] Philip R. Cohen, Adam Cheyer, Michelle Wang, Soon-Cheol Baeg, "An Open Agent Architecture," 1994 Spring Symposium Series: Software Agents, pp. 731-734, October 1994.

[8] Mark R. Cutkosky, Robert S. Engelmore, Richard E. Fikes, Michael R. Genesereth, and Thomas R. Gruber, "PACK: An Experiment in Integrating Concurrent Engineering Systems," Computer, pp. 28-37, January 1993.

[9] Mark d'Inverno and Michael Luck, "Understanding Autonomous Interaction," Proceedings of the 12th Conference on Artificial Intelligence, pp. 529-533, August 1996.

[10] T. Finin and R. Fritzson, "KQML-A Language and Protocol for Knowledge and Information Exchange," Proceedings of the 13th International Distributed Artificial Intelligence Workshop, pp. 127-136, USA, 1994.

[11] Michael R. Genesereth and Steven P. Ketchpel, "Software Agents," Communication of the ACM, pp. 48-53, Vol. 37, No. 7, July 1994.

[12] Christine Guilfoyle and Ellie Warnet, Intelligent Agents: the Revolution in Software, p. 214, Ovum Ltd, London, 1994.

[13] Jun Huang, N.R. Jennings and John Fox, "An Agent Architecture for Distributed Medical Care," In Michael J. Wooldridge and Nicholas R. Jennings, editor, Intelligent Agents, pp. 219-232, Springer-Verlag, Germany, 1995.

[14] Michael Kolb, "A Cooperation Languages," Proceedings of the First International Conference on Multi-Agent Systems, pp. 233-238, San Francisco, June 1995.

[15] Michael Luck and Mark d'Inverno, "A Formal Framework for Agency and Autonomy," Proceedings of the First International Conference on Multi-Agent Systems, pp. 254-260, June 1995.

[16] Murugappan Palaniappan, Nicole Yankelovich, George Fitzmaurice, Anne Loomis, Bernard Hann, James Coombs, and Norman Meyrowitz, "The Envoy Framework: An Open Architecture for Agents," ACM Transactions on Information Systems, Vol. 10, No. 3, pp. 233-264, July 1992.

[17] Mikhail Soutchanski and Eugenia Ternovskaia, "Logical Formalization of Concurrent Actions for Multi-Agent Systems," In Michael J. Wooldridge and Nicholas R. Jennings, editor, Intelligent Agents, pp. 129-144, Springer-Verlag, Germany, 1995.

[18] Michael Wooldridge and Nicholas R. Jennings, "Agent Theories, Architectures, and Languages: A Survey," In Michael J. Wooldridge and Nicholas R. Jennings, editor, Intelligent Agents, pp. 1-39, Springer-Verlag, Germany, 1995.

[19] T. Witting, N. R. Jennings and E. H. Mamdani, "ARCHON-A Framework for Intelligent Co-operation," IEE-BCS Journal of Intelligent Systems Engineering-Special Issue on Real-time Intelligent Systems in ESPRIT, 1994.

장 명 욱

1980년 고려대학교 전산과학과 졸업(학사)
1992년 한국과학기술원 전산학과 졸업(석사)
1992년~현재 한국전자통신연구소 인공지능연구실 연구원
관심분야: 에이전트 시스템, 패턴 인식, HCI

박 상 규

1982년 서울대학교 컴퓨터공학과 졸업(학사)
1984년 한국과학기술원 전산학과 졸업(석사)
1984년~1987년 대림산업 전산실 근무
1989년~현재 한국과학기술원 전산학과 박사과정
1987년~현재 한국전자통신연구소 인공지능연구실 선임연구원
관심분야: 에이전트 시스템, 정보 검색, HCI

이 광 로

1986년 Fukuoka 공업대학 전자기계공학과 졸업(학사)
1988년 Ritsumeikan 대학원 전기공학과 졸업(석사)
1994년~1995년 미국 SRI International(International Fellow)
1988년~현재 한국전자통신연구소 인공지능연구실 선임연구원
관심분야: 에이전트 시스템, 멀티미디어, 신경망

민 병 의

1982년 한양대학교 졸업(학사)
1984년 한국과학기술원 전기 및 전자공학과 졸업(석사)
1992년 한국과학기술원 전기 및 전자공학과 졸업(박사)
1984년~1987년 대림산업 기술연구소 근무
1987년~현재 한국전자통신연구소 인공지능연구실 실장
관심분야: 멀티미디어 시스템, 에이전트, 가상현실