계획 규칙의 통합을 통한 멀티 에이전트 시스템의 효율적인 작업 수행 방법

(A Method of Efficient Task Execution by Integrating Plan Rules in Multi-Agent Systems)

박 정 훈 † 최 중 민 ††

(Junghoon Park) (Joongmin Choi)

요 약 대부분의 에이전트는 그들이 생성될 때 자신이 수행하는 작업에 대한 계획을 갖고 있다. 이러한 에이전트들이 모여 멀티 에이전트 시스템을 이룰 경우 하나의 복잡한 작업을 수행하기 위해 미리 정의된 각 에이전트의 계획을 그대로 사용하게 되면 각 계획 규칙 사이의 연관성 등을 고려하지 못하기 때문에 시스템의 작업 처리 효율성이 떨어진다. 이를 해결하기 위해서는 에이전트가 갖고 있는 독자적인 계획을 통합하는 것이 요구된다. 이러한 계획의 통합은 미리 정의된 각 에이전트의 계획 사이의 관계를 파악하여 병렬적인 작업 수행을 가능하게 한다.

본 논문에서는 멀티 에이전트 시스템을 이루는 에이전트들이 자신의 작업 수행을 위한 계획을 갖고 있다는 가정 하에 하나의 큰 작업을 효율적으로 처리하기 위해 에이전트들의 계획을 네트워크 형태로 표현하고 이를 통합하여 작업을 수행하는 방법을 제시한다. 이 방법은 기존의 연구들이 다루고 있는 계획 순서 등과 관련된 에이전트 조정 문제를 자연스럽게 해결할 뿐만 아니라 다른 계획에 영향을 미치지 않는 각 에이전트의 독자적인 계획을 병렬로 수행시켜 작업 처리 시간을 단축시키는 효과를 낸다.

Abstract Most agents are associated with plan rules for their tasks when they are created. In a multi-agent system in which many agents are interacting, the direct use of predefined plan rules of each agents may slow down the system due to the lack of recognition of the relationship among plan rules of the agents. In order to overcome this, we need to analyze and integrate the agent's plan rules to facilitate concurrent rule execution.

This paper proposes a method that integrates and executes local plan rules of task agents in a multi-agent environment. The results of the integration are represented in a network structure. For domain task execution, a task agent collects other task agents' plan rules and builds an integrated domain network, which is exploited to achieve the goal. The agent problem solving by using the domain network enables not only the concurrent plan execution but the solution of coordination problems.

1. 서 론

에이전트는 복잡한 동적 환경에 존재하면서 주변 환경을 감지하며 지능을 갖고 필요에 따라 다른 에이전트와 통신을 통하여 사용자의 일을 대신해 주는 자율적인 프로그램이라 정의할 수 있다[1, 2]. 대부분의 에이전트는 그들이 고안될 때 자신이 수행해야 하는 작업을 달성하기 위한 연속적인 행동으로 정의되는 계획 규칙(plan rule)을 갖고 있으며 이에 기반하여 행동하게 된다. 일반적으로 하나의 독립된 에이전트는 간단한 작업을 처리하며 좀더 복잡하고 큰 작업을 처리하기 위해서는 에이전트들이 그룹화 되어 멀티 에이전트(multi agent) 시스템을 구성하게 된다.

이러한 멀티 에이전트 시스템에서는 에이전트들이 협동하여 효율적으로 하나의 작업을 완수하기 위해 각 에이전트를 통합하고 이들의 실행 순서 등을 조정

† 학생회원 : 한양대학교 전자계산학과
jhpark@cse.hanyang.ac.kr
†† 종신회원 : 한양대학교 전자계산학과 교수
jmchoi@cse.hanyang.ac.kr
논문접수 : 1999년 1월 26일
심사완료 : 2000년 6월 30일

(coordination)하는 것이 필요하다[3]. 이를 위해서는 각 에이전트가 갖고 있는 독자적인 계획을 통합하는 것이 요구된다. 이러한 계획의 통합은 미리 정의된 각 에이전트의 계획을 그대로 사용하는 것에 비해 좀더 병렬적인 작업 수행을 고려할 수 있으므로 시스템의 작업 처리 효율성을 증대시킬 수 있다. 멀티 에이전트 시스템에서는 각각의 에이전트가 도메인 전체에 대한 완벽한 지식을 갖고 있지 않으므로 공동의 문제를 해결하기 위하여 에이전트간 통신을 통한 정보 교환으로 상호 협력하게 되며 따라서 이런 정보를 내부의 지식 베이스(knowledge base)와 일관성을 가지도록 유지하는 것도 필요하다.

대부분의 기존 연구들은 에이전트들이 만들어질 때 계획을 가지고 있다고 생각하지 않았고, 따라서 시스템이 동작되면서 각 에이전트의 능력을 바탕으로 시스템에 효율적인 계획을 생성하거나 각각의 계획에 대한 순서(ordering) 등을 정하는 문제에 중심을 두었다. 멀티 에이전트 시스템에 관한 연구들도 이런 문제에 대한 것을 고려하고 있지 않다. 에이전트 시스템의 표준화 단체인 FIPA(Foundation for Intelligent Physical Agents)에서 제안한 에이전트 기반 구조를 이용하는 시스템들도 효율적인 작업 수행에 대한 연구 내용이 없는 상황이다. 하지만 대부분의 에이전트는 그들이 생성될 때 자신이 수행하게 될 작업에 대한 계획을 가지고 있기 때문에 이러한 상황에서는 효율적인 작업 수행을 위하여 각각의 에이전트가 갖고 있는 계획을 유기적으로 통합하여 하나의 계획을 만들어내는 것이 중요하다고 할 수 있다.

본 논문에서는 멀티 에이전트 시스템을 구성하는 에이전트들이 자신의 작업 수행을 위한 계획을 갖고 있다는 가정하에 하나의 큰 작업을 효율적으로 처리하기 위해 에이전트들의 계획을 네트워크 형태로 표현하고 이를 통합하여 작업을 수행하는 방법을 제시한다. 이 방법은 기존의 연구들이 다루고 있는 계획의 순서 등과 관련된 조정 문제를 자연스럽게 해결할 뿐만 아니라 다른 계획에 영향을 미치지 않는 각 에이전트의 독자적인 계획이 자동으로 병렬로 수행되는 것을 가능하게 하므로 작업 처리 시간을 단축시키는 장점을 지니게 된다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대하여 살펴보고 3장에서는 계획 규칙을 정의하고 각 에이전트의 계획 규칙을 네트워크 형태로 변환한 후 이를 통합하는 방법을 제시한다. 4장에서는 앞서 제시한 내용을 해외 출장 시나리오에 적용시킨 예를 보이고 5장에서 실험을 통하여 단일 네트워크로 통합된 계획 규칙에 의한 작업 수행의 효율성을 알아본다. 마지막으로 6장에서 결론을 맺고 향후 연구 과제에 대해서 언급한다.

2. 관련 연구

2.1 멀티 에이전트 계획 방법

Barber[4] 등은 첫 번째 계획 단계에서는 에이전트의 목적을 다루고 두 번째 계획 단계에서는 목적을 달성하기 위해 사용되는 연산자를 다루는 계획 방법을 제안하였다. 목적 지향 계획 단계에서는 복잡한 목적을 간단하면서도 의미 있는 작은 세부 계획으로 나누며 에이전트 사이의 조정을 다룬다. 여기서는 에이전트가 갖고 있는 의도된 계획을 평가하여 이것을 달성할 수 있는 세부 계획을 선택하게 된다. 연산자 지향 계획 단계에서는 목적과 현재 상태, 그리고 이용 가능한 연산자를 입력으로 받아서 목적 달성을 위해 사용할 연산자들을 선택하게 된다. 연산자는 행동으로 구성되어 있으며 에이전트의 역할을 나타내는 autonomy level(AL)에 의해 사전에 여과되어진다.

Ephrati[5, 6] 등은 멀티 에이전트 시스템의 계획을 위해 하나의 큰 문제를 분할해서 해결하는 방법에 대한 연구를 하였다. 이들의 기본적인 생각은 각각의 세부 계획은 전체 계획을 위한 탐색에 사용되는 휴리스틱 함수를 도출하는데 도움을 준다는데 있다. 계획 방식은 이미 존재하는 세부 목적을 사용하여 세부 계획을 도출하고 이를 바탕으로 전체 계획을 작성하게 된다. 그러므로 먼저 전체 목적을 세부 목적으로 나누고 각 에이전트가 이 세부 목적을 해결한다. 그런 다음 서로 다른 에이전트의 세부 목적 해결에 대한 세부 계획은 최종적으로 단일화된 유효한 전체 계획을 구성하기 위해 병합된다. 이 방식은 각 문제 해결을 여러 에이전트들이 병렬적으로 해결하므로 계획기에 의한 중앙식 계획 생성보다 계획을 생성하는데 걸리는 시간이 짧은 장점이 있다.

Georgeff[7]도 Ephrati 등과 비슷한 연구를 수행하였다. Georgeff는 문제 해결을 위한 멀티 에이전트 계획을 생성하기 위해 독립적으로 계획을 생성한 후 이를 합성하여 하나의 동기화 된 계획을 만들어 내는 방법을 제안하였는데 이를 위해 단일 에이전트 계획에 적절한 통신 행위를 삽입함으로써 에이전트들의 행동을 동기화하여 유해한 상호작용을 피할 수 있도록 하였다[8].

2.2 블랙보드 시스템

에이전트 기반의 블랙보드 시스템은 블랙보드와 에이전트, 그리고 제어 메커니즘으로 구성되어 있다. 블랙보드는 데이타베이스처럼 정보를 저장하고 있다. 에이전트

는 블랙보드의 상태를 살피면서 블랙보드에 정보를 게시하거나 이미 존재하는 정보를 변경하고 경우에 따라 필요한 정보를 읽어 들인다. 이 시스템에서 에이전트사이의 정보 교환은 오직 블랙보드를 통해서만 이루어진다. 제어 메커니즘은 에이전트의 블랙보드 이용 스케줄링과 블랙보드의 정보 유지 방법 등을 나타낸다[9]. 블랙보드 시스템은 인공지능 분야에서 문제 해결 방법으로 많이 이용되었는데 일반적으로 시스템을 구성하는 에이전트들이 정보 교환을 위해 블랙보드라는 하나의 공동 공간을 이용하기 때문에 에이전트간 직접적인 정보 교환에 비해 심각한 병목 현상을 유발하며 모든 에이전트들이 공통의 도메인에 대한 이해를 갖고 있어야 한다는 단점이 있다.

2.3 FIPA 에이전트 기반 구조

FIPA는 에이전트 표준화 단체로서 에이전트와 관련된 여러 가지 표준을 제시하고 있다. 제시한 표준중의 하나가 에이전트 관리[10]에 관한 것으로써 여기에는 에이전트 관리 참조 모델이 제안되었는데 이는 그림 1과 같이 에이전트 플랫폼(Agent Platform, AP)을 기반으로 시스템을 구성하고 있다.

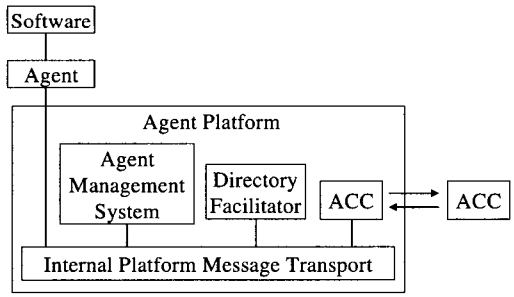


그림 1 FIPA의 에이전트 관리 참조 모델

AP에는 AMS(Agent Management System)와 DF(Directory Facilitator), 그리고 ACC(Agent Communication Channel) 등의 구성요소가 포함되어 있다. AMS는 AP내의 에이전트 등록 및 제거, 일시정지와 재시작 등의 전반적인 에이전트 관리 기능을 수행한다. DF는 각 AP내의 에이전트들의 제공 서비스에 대한 정보를 가지며 이를 이용하고자 하는 다른 에이전트에 제공하는 기능을 한다. ACC는 AP내부의 에이전트간 메시지 전송과 AP간의 통신을 지원하는 역할을 담당한다.

계획을 갖고 있는 에이전트들이 AP를 기반으로 멀티 에이전트 시스템을 이룰 경우 한 에이전트는 스스로 해결할 수 없는 문제를 DF, AMS, ACC등을 이용하여 다른 에이전트의 협조를 얻어 처리해 나간다. 이를 위해 먼저 DF에게 처리를 원하는 작업을 수행할 수 있는 에이전트에 대한 정보를 구한다. 그런 다음 AMS에게 해당 에이전트의 현재 상태에 대한 질의를 통하여 해당 에이전트가 동작 중인지 확인한다. 만약 동작중이 아닐 경우 AMS는 해당 에이전트를 호출하여 동작하게 만든다. 여기까지 완료되면 해당 에이전트에 문제 해결에 대한 메시지를 보내서 처리하게 만들고 이의 결과를 돌려받게 된다. AP 기반 멀티 에이전트 시스템에서는 이와 같은 방식으로 구성 에이전트들이 작업을 수행해 나가게 된다.

FIPA에서 제안한 에이전트 기반 구조를 이용하는 시스템들은 현재 병렬 수행에 대한 고려가 없다. 일련의 계획 규칙을 갖고 있는 에이전트는 계획을 수행하기 위해 DF, AMS, ACC 등을 계속적으로 이용하면서 계획을 하나씩 순차적으로 수행해 나가게 된다. 이것은 시스템내의 에이전트간 많은 통신을 요구하며 정의된 계획의 순차적인 수행이기 때문에 속도 측면에서도 단점이 존재한다. 그러므로 FIPA 에이전트 기반 구조하에서 병렬로 계획을 수행할 수 있는 방법을 추가한다면 속도 측면에서 많은 향상을 가져올 수 있을 것이다.

3. 계획 규칙과 네트워크 표현

에이전트가 처음 생성될 때 자신에게 주어진 작업을 완수하기 위한 계획을 갖고 있다고 가정하면 멀티 에이전트 시스템에서의 중요한 이슈중의 하나는 어떻게 각 에이전트의 계획을 통합하여 작업 수행이 효율적으로 이루어지도록 하는가가 된다.

본 논문에서는 멀티 에이전트 시스템에서의 효율적인 작업 수행을 위해 각 에이전트의 계획을 이루는 계획 규칙을 정의하고 이를 네트워크로 변환한 후 통합하여 하나의 계획으로 만들게 된다. 이렇게 만들어진 계획은 기존의 계획 관련 연구들이 중심적으로 다루는 계획의 순서와 관련된 조정 문제를 쉽게 해결하게 된다. 또한 네트워크로 통합된 계획을 이용하면 추가적인 노력 없이도 계획의 병렬 수행을 가능하게 하는 장점도 지니게 된다.

3.1 계획 규칙 정의

계획은 어떤 목적을 달성하기 위해 필요한 순서화 된 행동들의 집합으로 볼 수 있다. 계획을 구성하며 실행되는 행동을 나타내는 것이 계획 규칙이며 에이전트는 이런 계획 규칙의 수행을 통하여 그들의 목적을 달성하게 된다. 본 논문에서의 계획 규칙은 STRIPS(STanford Research Institute Problem Solver) 언어와 유사한 형

태로 표현한다[11]. STRIPS 연산자는 기본적으로 에이전트가 취하는 행동을 나타내는 행위 묘사자(action descriptor), 연산의 수행 전에 갖추어져야할 조건을 나타내는 전제조건(precondition), 연산 수행 이후의 변화를 나타내는 효과(effect) 등의 세 가지 요소로 구성된다. 하나의 연산자가 하나의 행동을 표현하고 이를 계획 규칙이라 하면 계획은 여러 개의 연산자로 구성되어 있다고 볼 수 있다. 본 논문에서 정의하는 연산자의 일반적인 형태는 다음과 같다.

OP(ACTION : Function expr,
PRECOND : Predicate expr,
EFFECT : Function expr)

연산자에서 *OP*는 다음에 나오는 괄호로 묶인 표현이 계획 규칙임을 나타낸다. *ACTION* 부분은 에이전트가 수행하는 행동을 나타내며 여러 가지 형태의 결과가 도출되기 때문에 함수(function) 형태의 표현을 취한다. 에이전트는 *ACTION*에 명시된 행동을 자신이 처리할 수 있으면 내부에서 수행하지만 그렇지 못할 경우엔 다른 에이전트에게 행동을 수행하여 결과를 넘겨줄 것을 요구하게 된다. *PRECOND* 부분은 *ACTION*이 수행되기 위한 전제 조건을 나타내며 모든 조건이 만족되었을 경우에만 에이전트가 *ACTION*을 수행하게 된다. 본 연구에서 이 부분은 확장된 술어를 사용하여 표현한다. 사용된 술어는 exist로 파라미터로 요구하는 믿음이 모두 존재하면 참(true)을, 그렇지 않으면 거짓(false)을 되돌린다. *EFFECT* 부분은 조건이 만족되어 에이전트가 *ACTION* 부분을 수행한 후 만들게 되는 결과를 나타낸다. 본 연구에서 *ACTION*의 결과에 의한 *EFFECT*는 전부 믿음을 만드는 과정이기 때문에 함수 표현식을 취하며 사용되는 함수는 MakeBelief이다. 이 함수는 파라미터를 믿음으로 만들어 준다.

3.2 계획 규칙의 종류

앞서 정의한 계획 규칙은 에이전트가 직접 처리할 수 있느냐에 따라 기본(elementary) 계획 규칙과 추상(abstract) 계획 규칙으로 나누어진다. 기본 계획 규칙은 *EFFECT* 부분이 MakeBelief 함수로 정의된다. 이 경우 에이전트는 계획 규칙을 자신이 직접 내부적으로 수행하고 새로운 결과를 만들게 된다. 추상 계획 규칙의 경우는 *EFFECT* 부분이 *None*으로 되어있다. 이와 같은 경우 에이전트는 계획 규칙을 직접 수행하여 완료할 수 없고 실제로 이 계획 규칙을 수행할 수 있는 다른 에이전트에게 수행을 요구하고 그 결과를 돌려 받아서 작업을 진행하게 된다.

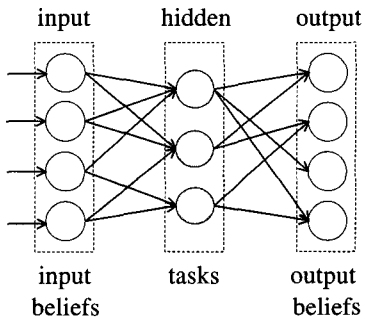


그림 2 에이전트 수행의 네트워크 표현

3.3 계획 규칙의 네트워크 표현

멀티 에이전트 환경에서 에이전트는 크게 작업 처리를 요구하는 클라이언트 역할의 에이전트와 요구받은 작업을 수행하는 서버 역할의 에이전트로 나눌 수 있다. 이 경우 클라이언트 에이전트는 서버 에이전트의 작업 수행에 필요한 정보 혹은 믿음을 넘겨주고 결과로써 서버 에이전트로부터 새로운 정보나 믿음 등을 돌려 받게 된다. 이는 그림 2와 같은 네트워크 형태로 표현될 수 있는데 서버측 에이전트로 넘겨지는 파라미터는 입력 계층(input layer), 서버측 에이전트가 실행하는 타스크는 히든 계층(hidden layer), 서버측 에이전트가 만들어 내는 결과는 출력 계층(output layer)으로 볼 수 있다[표 1].

3.3.1 형식화된 표현

이제 3.1에서 정의한 계획 규칙의 네트워크로의 표현을 형식화하여 살펴보도록 한다. 먼저 계획은 계획 규칙들로 구성되며 계획은 $\mathbb{P}$로, 계획 규칙은 *OP* 혹은 < $\mathbb{C}$, $\mathbb{A}$, $\mathbb{E}$ >로 표기한다. 계획 규칙 < $\mathbb{C}$, $\mathbb{A}$, $\mathbb{E}$ >에서 각각은 다음을 의미한다.

표 1 계획 규칙과 네트워크 계층의 대응 관계

계획 규칙	실제 대응값	네트워크
PRECOND	Input beliefs	Input layer
ACTION	Tasks	Hidden layer
EFFECT	Output beliefs	Output layer

- $\mathbb{C}$ = { c_1, c_2, ..., c_l }:계획 규칙의 *PRECOND*에 해당하며 전제조건 집합을 나타낸다.
- $\mathbb{A}$ = { a_1, a_2, ..., a_m }:계획 규칙의 *ACTION*에 해당하며 행동 집합을 나타낸다.
- $\mathbb{E}$ = { e_1, e_2, ..., e_n }:계획 규칙의 *EFFECT*에 해당

하며 효과 집합을 나타낸다.

하나의 계획 규칙에서 행동 부분은 함수에 의해 표현되는 하나의 행동으로 표현된다. 그러므로 **A**는 단순히 a로 표기할 수도 있다.

계획 규칙과 유사한 형태로 네트워크를 표현하면 < **I**, **H**, **O** >의 형태를 갖게 되며 각각은 다음과 같은 의미를 지니게 된다.

- **I** = { i_1, i_2, ..., i_l }:네트워크의 입력 계층에 해당하며 각 입력 노드를 의미한다.
- **H** = { h_1, h_2, ..., h_m }:네트워크의 히든 계층에 해당하며 각 타스크 노드를 의미한다.
- **O** = { o_1, o_2, ..., o_n }:네트워크의 출력 계층에 해당하며 각 출력 노드를 의미한다.

만약 'A↦B' 식에서 기호 ↦이 A라는 조건으로 B가 실행됨을 의미하고, 'C↣D'에서 기호 ↣이 C의 결과로 D가 도출된다는 의미를 갖는다고 하면 계획 규칙의 *ACTION*에 해당하는 **A**내의 행동과 네트워크에서 히든 계층 **H**에 있는 하나의 작업이 수행되기 위한 조건과 수행후의 결과는 각각 다음과 같이 나타낼 수 있다.

$$c_1, c_2, ..., c_l \mapsto a_k \rightarrowtail e_1, e_2, ..., e_n$$

$$i_1, i_2, ..., i_l \mapsto h_k \rightarrowtail o_1, o_2, ..., o_n$$

그러므로 기호 ≈이 대응 관계를 의미한다고 할 때 계획 규칙이 네트워크로 표현된다는 것은 계획 규칙과 네트워크 사이에 다음의 대응 관계가 성립한다는 것을 의미한다.

$$\mathbf{C} \approx \mathbf{I} \Leftrightarrow \{ c_1 \approx i_1, c_2 \approx i_2, ..., c_l \approx i_l \}$$
$$\mathbf{A} \approx \mathbf{H} \Leftrightarrow \{ a_1 \approx h_1, a_2 \approx h_2, ..., a_m \approx h_m \}$$
$$\mathbf{E} \approx \mathbf{O} \Leftrightarrow \{ e_1 \approx o_1, e_2 \approx o_2, ..., e_n \approx o_n \}$$

계획 규칙으로 구성된 계획을 네트워크로 표현하게되면 이와 같은 관계에 의하여 초기 상태이후 **I**, **H**, **O**가 반복적으로 나타나다가 목적 상태에 도달하게 된다.

앞서 정의한 계획 규칙을 네트워크로 변환하는 방법은 간단하다. 의미적으로 *PRECOND* **C**의 믿음들을 네트워크 입력 계층 **I**의 하나의 노드 i에 일대일 대응시키고 *ACTION* **A**에서 수행되는 작업을 히든 계층 **H**의 하나의 노드 h가 되도록 하면 된다. 그리고 *EFFECT* **E**의 결과들은 각각 출력 계층 **O**의 하나의 노드 o에 대응되도록 하면 된다[그림 3]. 이러한 계획 규칙과 신경망 구조의 대응 관계로부터 각 에이전트의 계획 규칙은 네트워크로 표현될 수 있다.

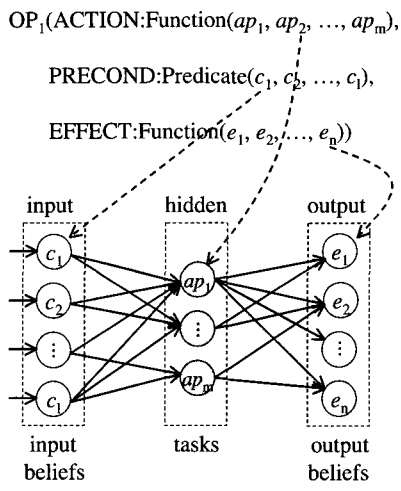


그림 3 계획 규칙과 네트워크 계층의 대응 관계

3.4 계획의 병렬 수행 조건

이번 절에서는 [12], [13], [14], [15] 등에서 사용한 표현 방식과 유사한 형식으로 계획의 병렬 수행에 대한 것을 형식화시킴으로써 계획 병합에 대한 것을 개념적으로 알아본다.

3.1에서 정의한 것처럼 계획(ℙ)이 여러 연산자(α)로 구성되어 있다하고 하면 이는 다음처럼 표기할 수 있다.

$$\mathbb{P} = \{ \alpha_1, \alpha_2, ..., \alpha_n \}$$

이와 같이 표기할 때 계획 병합은 각 계획을 구성하는 연산자의 병합으로도 볼 수 있다. 앞서 정의된 연산자 정의에서 연산자 α가 수행되기 위한 전제조건을 *Preconds*(α)로, 연산자 α의 수행 후 효과를 *Effects*(α)로 표현하고 이들의 관계를 3.3에서 정의한 ↦ 와 ↣기호를 이용하여 나타내면 전제조건과 효과 및 연산자의 관계는 다음과 같다.

$$Preconds(\alpha) \mapsto \alpha \rightarrowtail Effects(\alpha)$$

일반적으로 계획을 다루는 시스템에서는 그들의 실행 순서가 정해진다. 기호 <이 실행 순서를 나타낸다고 하면 {$\alpha_1 < \alpha_2$}는 연산자가 α_1, α_2 순서로 실행됨을 의미하게 된다. 만약 계획 ℙ가 $\mathbb{P}_1$={α_1, α_4, α_6}, $\mathbb{P}_2$={α_3, α_5, α_9}, $\mathbb{P}_3$={α_2, α_7, α_8}로 구성된 세부 계획으로 이루어져

있으며 각 연산자의 첨자가 그들의 실행 순서를 의미한다면 계획 $\mathbb{P}$를 구성하는 연산자의 실행 순서는 $\{\alpha_1 < \alpha_2 < \alpha_3 < \alpha_4 < \alpha_5 < \alpha_6 < \alpha_7 < \alpha_8 < \alpha_9\}$와 같이 α_1부터 α_9까지의 순차적인 순서를 갖게 된다. 여기에는 계획, 즉 연산자의 병렬 수행에 대한 개념이 없다. 그러나 계획을 구성하는 연산자는 병렬로 수행이 가능한 경우가 많다. 연산자들이 병렬로 수행되기 위해서는 연산자들의 전제조건이 만족된 상태에서 연산자들 사이에 의존성이 존재하지 않아야 한다. 연산자 사이의 의존성은 한 연산자의 *Effects*()가 다른 연산자의 *Preconds*()일 때 존재하게 된다. 그러므로 연산자 α_j과 α_k는 다음 세 가지 조건을 만족할 때 병렬로 수행이 가능하다고 할 수 있다.

1. $\forall \alpha_i \in \mathbb{P}$, $\{ \alpha_i < \alpha_j \}$, $\{ \alpha_i < \alpha_k \}$를 만족하는 α_i, α_j, α_k에 대해서

   $Preconds(\alpha_j) \in \cup Effects(\alpha_i)$이며 $Preconds(\alpha_k) \in \cup Effects(\alpha_i)$

2. $\forall p_j \in Preconds(\alpha_j)$에 대해서 $p_j \notin Effects(\alpha_k)$ or $\{ \alpha_k \prec \alpha_j \}$

3. $\forall p_k \in Preconds(\alpha_k)$에 대해서 $p_k \notin Effects(\alpha_j)$ or $\{ \alpha_j \prec \alpha_k \}$

여기서 조건 1은 전제조건이 만족된 상태를 의미하고 조건 2와 3은 병렬 수행하고자 하는 연산자간 상호 의존성이 존재하지 않음을 의미하는 것으로써 Knoblock [16]의 자료 의존성 및 병렬 수행 계획의 세 번째 종류에 대응된다고 볼 수 있다.

앞서의 $\mathbb{P}_1$, $\mathbb{P}_2$, $\mathbb{P}_3$로 구성된 계획 $\mathbb{P}$에서 (α_2, α_3), $(\alpha_5, \alpha_6, \alpha_7)$이 위의 조건을 만족하여 병렬 수행이 가능한 연산자들이라면 전체적인 수행 순서는 $\{\alpha_1 < (\alpha_2, \alpha_3) < \alpha_4 < (\alpha_5, \alpha_6, \alpha_7) < \alpha_8 < \alpha_9\}$와 같게 된다. 만약 모든 연산자가 수행되는데 걸리는 시간이 동일하다고 가정하고 이를 T로 나타내면 앞서의 병렬 수행 개념이 들어간 순서에 의한 수행은 순차적인 순서에 의한 수행이 $9T$라는 시간이 소요되는 것과 달리 $6T$라는 시간이 소요되므로 3T만큼의 수행 시간을 단축할 수 있게 된다.

3.5 계획 규칙의 단일 네트워크로의 통합

계획 $\mathbb{P}$는 여러 개의 계획 규칙으로 구성되어 있으며 3.1과 3.3절에서 정의한 형태로 표현되므로 계획 $\mathbb{P}$는 다음처럼 나타낼 수 있다.

$$\mathbb{P} = \cup OP_i = \cup\{< \mathbb{C}_i, \mathbb{A}_i, \mathbb{E}_i >\}$$

계획 $\mathbb{P}$를 이루는 각 계획 규칙은 3.3절의 방법에 의해 네트워크 형태로 변환된다. 본 연구에서 기반을 두고 있는 멀티 에이전트 시스템에서 각 에이전트들은 자신이 수행하는 작업에 대한 계획을 갖고 있다고 가정하므로 하나의 에이전트가 병렬 수행을 위해 필요한 계획을 모으고 이를 네트워크로 표현한 후 하나로 통합하게 된다. 그러므로 계획 $\mathbb{P}$는 다시 다음처럼 나타낼 수 있다.

$$\mathbb{P} = \cup \mathbb{P}_i = \cup(\cup OP_i) = \cup(\cup\{< \mathbb{C}_i, \mathbb{A}_i, \mathbb{E}_i >\}) = \cup\{< \mathbb{I}_i, \mathbb{H}_i, \mathbb{O}_i >\}$$

계획 규칙을 네트워크 형태로 바꾼 형태를 $< \mathbb{I}, \mathbb{H}, \mathbb{O} >$로 표현하므로 이의 통합은 $\mathbb{I}$, $\mathbb{H}$, $\mathbb{O}$가 반복적으로 나타나게 된다. 하나의 네트워크에서 $\mathbb{O}$는 다음 네트워크의 $\mathbb{I}$에 사용되므로 실제로는 첫 $\mathbb{I}$이후 $\mathbb{H}$, ($\mathbb{O}$, $\mathbb{I}$)가 연속적으로 나타나다가 $\mathbb{O}$로 끝나게 된다.

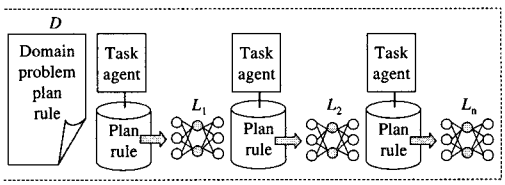


그림 4 도메인 문제에 대한 시스템 표현

에이전트들의 모든 계획 규칙을 네트워크 형태로 표현한다면 특정 도메인에서의 멀티 에이전트 시스템은 그림 4와 같은 형태를 갖게 될 것이다. 그림 4에서 도메인 계획 규칙의 네트워크 표현인 D와 각 작업 에이전트의 계획 규칙을 네트워크로 표현한 L_1, L_2, ..., L_n을 3.3절에서 정의한 병렬 수행 조건에 따라 통합하면 최종적으로 하나의 도메인 네트워크를 구성할 수 있다. 기호 $\otimes$이 통합을 의미한다면 이는 다음과 같은 식으로 나타낼 수 있다.

$$Domain\ Network = D \otimes L_1 \otimes L_2 \otimes \ldots \otimes L_n$$

그림에서 각 에이전트의 계획 규칙을 표현한 네트워크를 통합하여 도메인 네트워크로 구성하려면, 하나의 네트워크에서 도출된 믿음 노드를 입력 믿음 노드로 받아들일 수 있는 다른 모든 네트워크에 연결해 나가면 된다.

이처럼 도메인 문제 해결에 대한 계획 규칙들을 각각의 네트워크로 표현한 후 이를 통합하여 하나의 도메인 네트워크를 만들면 계획 규칙 대신에 구성된 네트워크로 도메인 문제를 해결 할 수 있다. 이를 통한 작업 진행은 미리 정의된 계획 규칙에 의한 순차적인 행동의 수행과 달리 일반적인 계획 규칙이 자연스럽게 병렬로 수행되는 것을 가능하게 한다.

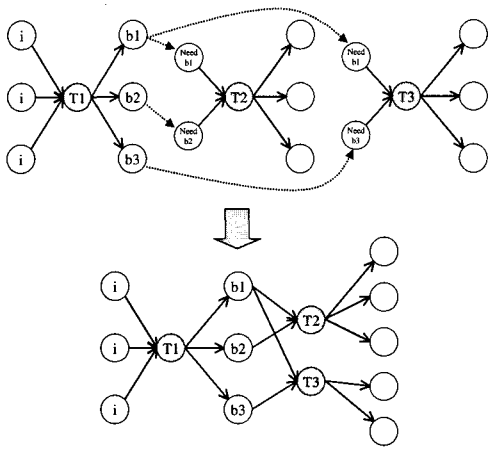


그림 5 네트워크 통합 예

그림 5에 표현된 것과 같은 관계를 갖는 T1, T2, T3의 네트워크 표현이 있고 {T1 < T2}, {T1 < T3}라 하자. 또한 $\exists p_1 \in$ *Precond*(T3)에 대해서 $p_1 \notin$ *Effects*(T2)이고 $\exists p_2 \in$ *Precond*(T2)에 대해서 $p_2 \notin$ *Effects*(T3)이며 *Precond*(T2) ∈ *Effects*(T1), *Precond*(T3) ∈ *Effects*(T1)를 만족한다하면 T2와 T3는 3.3절에서 언급한 두 연산자의 병렬 수행 조건의 세 가지를 모두 만족시키므로 병렬 수행이 가능하다. 즉, {T1 < (T2,T3)}의 의미를 갖게 된다.

4. 시나리오

이번 장에서는 해외 출장 시나리오에 대한 에이전트의 계획을 네트워크 형태로 만들고 이것을 하나의 도메인 네트워크로 통합하는 것을 보임으로써 3장에서 설명한 내용이 어떻게 구현되고 실행되는지를 살펴본다.

4.1 해외 출장 시스템 구조

도메인은 해외 출장에 한정되며 해외 출장 시스템의 전체적인 구조는 그림 6과 같다. DF와 AMS는 FIPA에서 제안한 에이전트 기반구조[10]에 있는 것으로 이의 기능은 2.3절에서 언급했다. 작업 에이전트에 해당하는 교통예약, 숙박예약, 비자/여권 발급, 개인정보관리 및 출장비용계산 에이전트는 각 에이전트 이름에 해당하는 작업을 수행한다. 본 시스템에서 중심적인 역할을 하는 에이전트는 인터페이스 에이전트로 사용자가 요구한 작업을 해결하기 위해 필요한 작업 에이전트들의 계획을 모으고 이를 네트워크 형태로 변환한 후 통합한다. 사용자가 요구한 작업을 달성하게 된다. 인터페이스 에이전트는 이런 과정에서 사용자로부터 추가적인 정보 입력이 필요할 때 이를 입력받는 기능도 하게 된다.

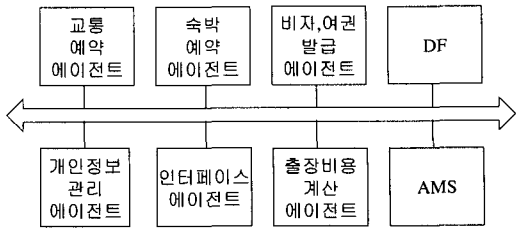


그림 6 해외 출장 시스템

4.2 에이전트 계획

각 작업 에이전트는 자신이 수행하는 작업에 대한 계획을 갖고 있다. 이 계획은 3.1절에서 정의한 계획 규칙의 집합으로 표현되며 인터페이스 에이전트의 계획 규칙은 다음과 같다.

1. *OP*(*ACTION*: GetUserData(이름, 주민등록번호, 주소,..., 연락처), *PRECOND*: None, *EFFECT*: MakeBeliefs(이름, 주민등록번호, 주소,..., 연락처))
2. *OP*(*ACTION*: MakeVisa(이름, 주민등록번호, 주소,..., 여행목적), *PRECOND*: exist(이름, 주민등록번호, 주소,..., 여행목적), *EFFECT*: None)
3. *OP*(*ACTION*: ReserTraf(이름, 주민등록번호, 주소,..., 은행잔고), *PRECOND*: exist(이름, 주민등록번호, 주소,..., 은행잔고), *EFFECT*: None)
4. *OP*(*ACTION*: ReserLodg(이름, 목적지,..., 은행잔고), *PRECOND*: exist(이름, 목적지,..., 은행잔고), *EFFECT*: None)
5. *OP*(*ACTION*: CompTotCost(여권발급수수료, ...), *PRECOND*: exist(여권발급수수료, ...), *EFFECT*: None)

각 계획 규칙에서 *ACTION*과 *EFFECT*에 사용되는 함수의 파라미터는 field명만 나타내었는데 실제 수행할 때는 'field=value' 형태를 갖는다. *PRECOND*의 exist는 파라미터에 해당하는 믿음이 모두 존재하면 참을, 그렇지 않을 경우 거짓을 되돌린다.

계획 규칙을 보면 1번의 경우 *PRECOND*가 None으로 되어있으며 *EFFECT*부분이 MakeBelief 함수로 되어 있다. 이것은 전제조건이 없는 기본 계획 규칙을 의미하며, 전제조건이 없으므로 계획이 바로 실행되어 사용자로부터 필요한 정보를 입력 받게된다. 계획 규칙 2, 3, 4, 5번은 *EFFECT*부분이 None으로 추상 계획 규칙을 나타내고 있다. 추상 계획 규칙은 에이전트

자신이 직업 해결할 수 없는 작업을 나타낸다. 이 경우 인터페이스 에이전트는 DF와 AMS를 이용하여 이를 수행하는 에이전트를 통해 문제를 해결하게 된다.

다른 작업 에이전트들도 인터페이스 에이전트와 같은 형태로 계획 규칙이 정의된다.

4.3 계획 수집과 네트워크로의 표현

사용자가 해외 출장 작업을 요구하면 인터페이스 에이전트는 먼저 해외 출장 작업 계획을 네트워크로 변환하게 된다. 계획 중에서 [1]번과 같은 기본 계획 규칙은 바로 네트워크 형태로 변환될 수 있지만 [2], [3], [4], [5]번과 같은 추상 계획 규칙의 경우에는 이를 해결할 수 있는 에이전트의 도움을 받아야 하므로, 각각을 수행할 수 있는 에이전트로부터 계획을 갖고 와서 이를 네트워크 형태로 변환하게 된다.

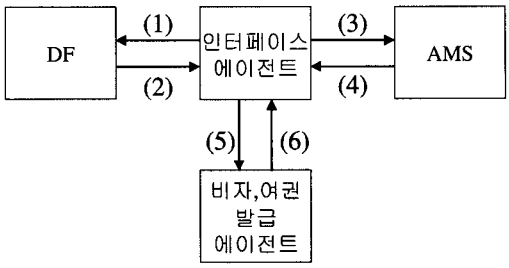


그림 7 인터페이스 에이전트의 작업 흐름

계획 규칙 [2]를 보게되면 *ACTION*부분의 함수가 MakeVisa인데 이것과 DF, AMS를 이용하여 해당 에이전트로부터 계획을 받게 된다. 먼저 인터페이스 에이전트는 DF에 KQML 메시지를 보내 MakeVisa를 처리할 수 있는 에이전트에 대해 묻게 된다. DF는 인터페이스 에이전트로부터 위와 같은 메시지를 받게 되면 해당 작업을 처리할 수 있는 에이전트를 결과로써 돌려주게 된다. 여기서는 비자/여권발급 에이전트가 이에 해당된다. 인터페이스 에이전트는 DF로부터 결과를 받아 MakeVisa를 처리할 수 있는 에이전트가 비자/여권 발급 에이전트라는 것을 알게되면 이 에이전트의 구동을 AMS에 요청하게 된다. AMS는 비자/여권발급 에이전트가 구동중이 아니라면 이를 구동시킨다. 마지막으로 인터페이스 에이전트는 비자/여권발급 작업 에이전트에 계획을 요구하여 돌려 받게 된다. 이 같은 작업 흐름은 그림 7에 나타나 있다. 이와 같은 방식으로 계획 규칙 [3], [4], [5]번을 해결할 수 있는 에이전트로부터 각각의 계획을 가져오게 되며 이를 네트워크 형태로 변환한다.

계획 규칙의 네트워크 형태로의 변환은 *PRECOND*의 exist에 요구되는 믿음을 입력 계층으로 만들고 *ACTION*에 있는 함수를 히든 계층으로 만든다. 그리고 *EFFECT*의 MakeBelief에 의해 만들어지는 믿음을 출력 계층으로 나타내면 된다. 이와 같은 방식으로 표현하면 해외 출장 계획은 그림 8과 같이 표현된다. 계획 규칙 [1]는 *PRECOND*가 없어서 바로 실행될 수 있기 때문에 입력 계층이 없다. 필요한 계획을 모으고 이를 네트워크 형태로 변환하여 그림 8과 같은 결과를 얻었다면 이를 통합하여 하나의 네트워크로 구성해야한다. 단일 네트워크로의 통합은 3.4절에서 설명한 것처럼 계획의 병렬 수행 조건을 고려하면서 한 네트워크의 출력 계층 노드를 이를 필요로 하는 다른 네트워크의 입력 계층 노드에 연결하면 된다. 그림 8에서 각 노드의 번호가 믿음의 인덱스라고 가정하면 입력 계층의 노드 번호는 조건으로 필요로 하는 믿음을 나타내며 출력 계층의 노드 번호는 결과로 만들어지는 믿음이 된다. 도메인 네트워크의 구성은 이 번호를 서로 연결함으로써 이루어진다. 이는 그림 9에 나타나있다. 통합된 단일 네트워크로부터 여권신청, 비자신청, 교통예약, 숙박예약 등의 작

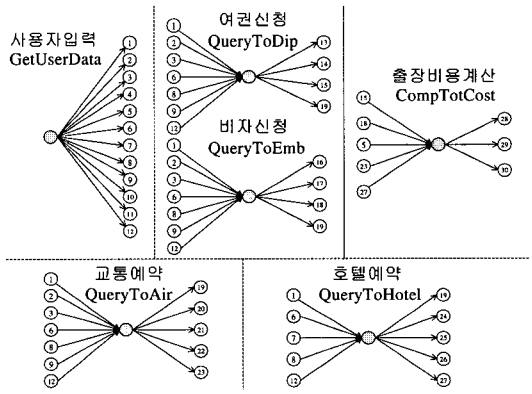


그림 8 해외 출장 계획 규칙의 네트워크 변환

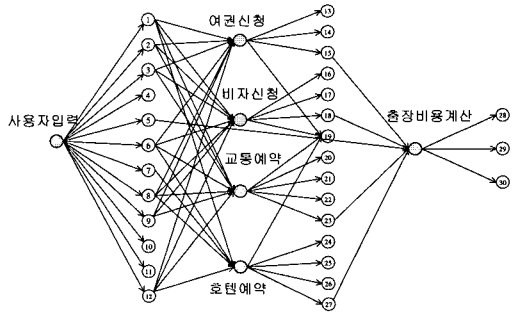


그림 9 해외 출장 계획의 최종 네트워크

업이 병렬 수행 조건을 만족하고 있음을 알 수 있다. 그러므로 이들 작업은 병렬적으로 수행된다. 결과적으로 해외 출장 계획은 {사용자입력 < (여권신청, 비자신청, 교통예약, 호텔예약) < 출장비용계산}과 같이 표현될 수 있다.

4.4 네트워크로 표현된 계획의 실행

네트워크 형태로 표현된 계획의 실행에 앞서 네트워크가 프로그램 상에서 어떻게 표현되는지 살펴보자. 계획의 네트워크 형태는 노드와 링크로 구성되는 리스트 자료형으로 표현한다. 네트워크의 입력과 출력 계층은 믿음을 의미하므로 하나의 믿음 리스트로 관리하고 히든 계층은 수행할 작업을 나타내는 믿음과는 다른 작업 리스트로 관리한다. 그러므로 각 리스트에서의 하나의 노드는 각각 믿음과 작업을 나타낸다. 수행할 작업은 그들의 전제조건으로 믿음 리스트에서 몇 개의 노드를 필요로 하는데 이는 전제조건 리스트에 의해 표현된다. 전제조건 리스트는 작업 리스트에 있는 각 작업이 어떤 믿음을 필요로 하며 그 믿음이 지금 존재하는지에 대한 정보를 유지하고 있다.

계획 규칙이 네트워크로 변환된 다음 통합되어 단일 네트워크로 구성되고 이 네트워크가 위와 같은 구조로 표현이 되면 인터페이스 에이전트는 이를 바탕으로 네트워크로 표현된 계획을 실행하게 된다. 네트워크로 표현된 계획을 수행하는 인터페이스 에이전트는 그림 10과 같이 5개의 구성요소로 되어 있다.

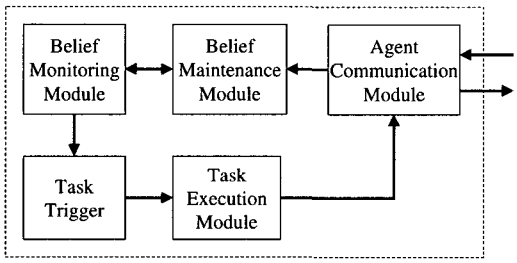


그림 10 인터페이스 에이전트 구조

믿음 유지 모듈(Belief Maintenance Module)은 표현된 네트워크에서 입력과 출력 계층에 해당하는 믿음 리스트를 관리한다. 믿음 유지 모듈은 새로운 믿음에 들어오면 새로운 노드를 만들어 할당하고 이를 리스트에 연결한다. 믿음이 무효화되면 해당 노드를 리스트에서 삭제하게 된다. 믿음 감시 모듈(Belief Monitoring Module)은 믿음 유지 모듈을 감시하면서 믿음의 추가나 삭제 등 리스트에 변화가 생기면 이를 작업 트리거에 알리게 된다. 작업 트리거(Task Trigger)는 각 작업의 전제조건을 감시하면서 만족되는 작업이 있을 경우 해당 작업의 수행을 작업 수행 모듈에 요구하게 된다. 작업 수행 모듈(Task Execution Module)은 작업 트리거의 요청이 있으면 해당 작업을 수행하는데 이는 에이전트 통신 모듈을 통해 해당 작업 에이전트에 작업 수행 요구 메시지를 보내는 것이 된다. 인터페이스 에이전트는 이와 같은 5개의 구성요소가 유기적으로 동작을 수행함으로써 작업을 진행하게 되며 더 이상 수행해야 할 작업이 리스트에 남아 있지 않으면 모든 작업이 완료된 것으로 판단하여 최종 결과를 표시하고 전체 작업을 종료하게 된다.

그림 9에 표현된 네트워크를 수행한다면 초기에 작업 리스트에는 6개의 작업이 존재하게 된다. 사용자의 입력을 받는 것은 전재 조건이 없으므로 바로 트리거 되어 작업이 수행되게 된다. 사용자의 입력이 들어와서 이를 믿음으로 만들게되면 믿음 감시 모듈에 의해 여권신청, 비자신청, 교통예약, 호텔예약 등의 작업이 수행 가능한 것으로 검색되어지며 바로 작업 트리거 모듈에 이의 실행을 요구하여 4가지 작업이 차례로 수행되게 된다. 4가지 작업이 완료되면 마지막 작업으로 출장비용 계산 작업이 수행되며 이의 종료는 더 이상 작업이 남아 있지 않는 것을 의미하므로 인터페이스 에이전트는 종료하게 된다. 이처럼 계획 규칙이 통합되어 단일 네트워크로 표현한 후 실행하게 되면 병렬 수행이 가능한 작업이 추가적인 노력 없이도 병렬로 수행되는 장점을 지니게 된다. 그림 11에 나타난 것처럼 계획 규칙을 그대로 이용하여 작업을 수행하는 것보다 시간적으로 많은 이득을 볼 수 있다.

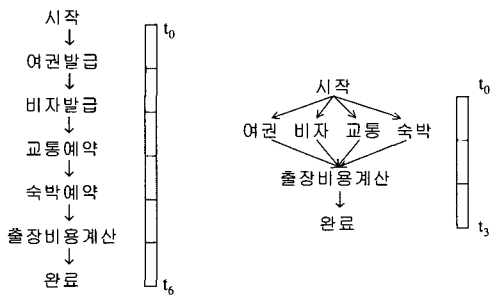
5. 실험 결과

이번 장에서는 실험을 통하여 본 논문에서 제안하는 계획 규칙의 단일 네트워크로의 통합에 의한 작업 수행 기법이 멀티 에이전트 시스템에서의 작업 수행에 어느 정도 속도 향상을 가져오는지 알아본다. 실험을 통하지 않고도 네트워크 표현에 의한 병렬 수행 방법이 속도 향상을 가져올 것을 예상할 수 있지만 실험을 통하여 정확히 알아보고자 한다.

5.1 실험 모델 및 방법

본 논문에서는 두 가지 모델에 대하여 실험을 수행하여 그 결과를 비교하였다. 사용된 모델 중 하나는 FIPA 표준안에 기반 한 직렬 수행 모델로써 하나의 큰 작업을 완성하기 위해 시스템을 이루는 작업 에이전트들을

계획 규칙 순서대로 하나씩 이용하게 된다. 다른 하나는 병렬 수행 모델로써 역시 직렬 수행 모델과 마찬가지로 FIPA 표준안에 기반을 두고 있지만 작업 진행에 있어서 본 연구에서 제안하는 방식을 이용하게 된다. 먼저 사용하고자 하는 작업 에이전트들의 계획을 모으고 이를 네트워크로 표현한 다음 단일 네트워크로 통합한 후 작업을 진행하게 된다.



[일반 계획] [네트워크 표현]

그림 11 일반 계획과 네트워크 표현에 의한 계획 수행 소요 시간

실험 방법은 두 가지 모델에 대하여 각각 10, 20, 30, 50개의 작업 에이전트로 이루어진 멀티 에이전트 시스템을 구성하여 작업 수행 속도를 비교하였다. 직렬 수행 모델에서는 작업 에이전트들이 한번씩 순차적으로 수행되도록 멀티 에이전트 시스템을 구성하였으며 병렬 수행 모델에서는 추가적으로 작업 에이전트의 0%, 10%, 20%, 30%, 50%가 병렬로 수행되도록 시스템을 구성하였다. 프로그램은 C언어를 사용하여 구현되었으며 실험은 SunOS 5.5.1이 설치된 Sun Ultrasparc에서 이루어졌다. 작업 수행에 소요되는 시간은 cpu time으로 계산하였고 각 실험을 5회 반복 실행하여 나온 시간의 평균으로 소요 시간을 계산하였다.

5.2 실험에서의 가정

본 실험을 수행하는데 있어서 몇 가지 가정을 하였는데 이에 대한 것을 살펴본다.

가정 1. 두 모델에서 DF와 AMS를 이용하는데 소요되는 시간은 동일하다.

실험에 이용되는 두 모델 모두 FIPA에서 제안한 에이전트 기반 구조에 바탕을 두고 있기 때문에 작업을 진행하기 위해서는 DF와 AMS를 이용하게 된다. 두 모델에 공통된 사항이므로 모델마다 소요 시간이 다르다면 정확한 실험이 수행되지 못한다. 그러므로 두 모델에서 이를 이용하는데 소요되는 시간이 동일하다고 가정하는 것이 필요하다.

가정 2. 모든 작업 에이전트는 자신의 작업을 수행하는데 소요되는 시간이 동일하다.

만약 작업 에이전트마다 작업 처리시간이 다르다면 어느 작업 에이전트들을 이용하여 시스템을 구성하느냐에 따라 실험 결과가 계속해서 다르게 나올 것이다. 그러므로 정확한 실험과 비교가 되기 위해서는 각 작업 에이전트의 작업 수행에 소요되는 시간은 동일해야 한다. 본 실험에서 작업 에이전트의 작업 수행 시간은 1 cpu time으로 통일하였다.

가정 3. 작업 에이전트간 상호 참조는 없다.

한 작업 에이전트가 자신의 작업을 처리하기 위해 다른 작업 에이전트를 이용하고 이것이 시스템에 계속적으로 나타나게 된다면 이 역시 시스템을 어떻게 구성하느냐에 따라 실험 결과가 다르게 나오게 된다. 그러므로 정확한 실험이 이루어지도록 작업 에이전트간 상호 참조는 없는 것으로 가정한다.

5.3 실험 결과

표 2는 직렬 수행 모델에서의 실험 결과를 나타내고 있다. 하나의 작업 에이전트가 작업을 수행하는데 걸리는 시간을 1 cpu time으로 할 때 작업 수와 작업 수행에 소요되는 시간은 거의 정비례 관계를 이루게 된다 [그림 12]. 작업 중에서 병렬 수행이 가능한 작업이 존재하여도 직렬로 수행되기 때문에 정비례 관계가 계속 유지된다.

표 2 직렬 수행 모델의 실험 결과

작업 에이전트 수	10	20	30	50
소요 시간	10.108	20.207	30.327	50.520

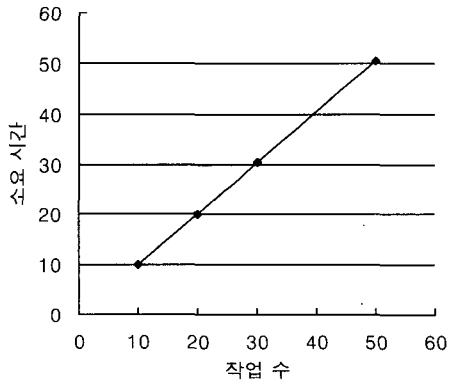


그림 12 직렬 수행에서의 작업 수와 소요 시간 관계

표 3 병렬 수행 모델의 실험 결과

	10	20	30	50
0%	10.125	20..241	30.373	50.623
10%	9.126	18.271	27.374	45.641
20%	8.117	16.215	24.355	40.707
30%	7.113	14.214	21.348	35.546
50%	5.105	10.196	15.299	25.913

표 4 병렬 수행에 의한 속도 향상 정도

	10	20	30	50
10%	9.715%	9.581%	9.737%	9.658%
20%	19.697%	19.756%	19.692%	19.424%
30%	29.630%	29.658%	29.607%	29.640%
50%	49.495%	49.542%	49.553%	48.707%

표 3과 4는 각각 병렬 수행 모델의 실험 결과 및 병렬 수행에 의한 속도 향상 정도를 나타내고 있다. 작업 수와 소요 시간은 직렬 수행 모델과 마찬가지로 비례 관계를 이루고 있다. 하지만 예상한 것처럼 병렬적으로 작업을 수행함으로 직렬 수행보다는 완만한 경사를 이루게 된다. 또한 전체 작업 중 병렬 수행이 가능한 작업의 수가 많을수록 더 완만한 경사를 이루게 되며 속도 향상 정도를 보면 병렬 수행 작업 수가 차지하는 비율만큼이 향상되는 것을 알 수 있다[그림 13]. 다만 병렬 작업이 가능한 작업 수가 없을 경우에는 직렬 수행 모델보다 시간이 더 소요되었는데 이유는 병렬 작업 수행 준비를 위한 계획 규칙의 통합이라는 추가적인 시간 소모가 있었지만 계획 규칙에 병렬로 수행 가능한 작업이 존재하지 않아서 본 연구에서 제안하는 방법의 효과를 살리지 못하기 때문이다.

6. 결론 및 향후 연구 과제

본 연구에서는 멀티 에이전트 시스템에서 효율적인 작업 수행을 위해 시스템을 구성하는 각 에이전트의 계획 규칙을 네트워크 형태로 표현하고 이를 통합하여 하나의 네트워크로 만든 다음 이를 통하여 작업을 수행하는 방법을 제안하였다. 이렇게 구성된 네트워크를 통한 작업 수행은 기존의 연구들이 다루고 있는 순서 문제 등과 관련된 조정에 관한 문제를 쉽게 해결한다. 또한 추가적인 노력 없이도 순차적인 계획 규칙 중 병렬 수행이 가능한 부분을 자동으로 병렬로 진행되게 하므로 최대 병렬 작업이 차지하는 비중만큼 작업 시간을 단축하는 장점을 지니게 된다.

향후 연구 과제로는 구성된 네트워크를 바탕으로 추론할 수 있는 메커니즘의 정립이 필요하다. 현재의 연구 내용에서는 작업 수행 중 이전 작업으로 되돌아가는 문제에 대한 고려가 없다. 이것은 작업 수행 중에 빈번히 일어날 수 있으므로 이에 대한 대책이 필요하다. 또한 에이전트가 존재하는 환경이 불확실한 경우가 많으므로 주변환경의 불확실성에 대처할 수 있는 방법에 대한 연구가 필요하다. 본 연구의 방법이 계획 규칙을 네트워크로 표현한 후 통합하여 계획을 수행하는 것이므로 네트워크 모델에 확률 개념을 추가하면 이에 대한 대비가 될 것이라 생각된다.

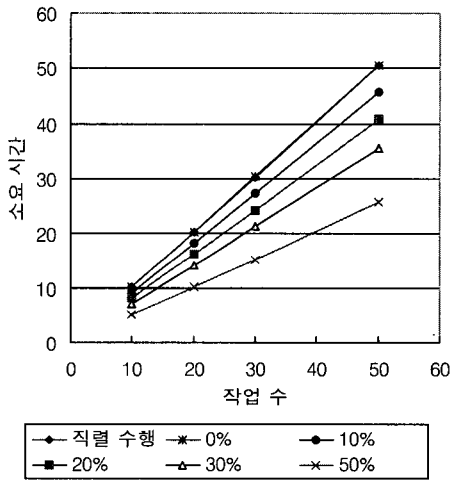


그림 13 병렬 수행에서의 작업 수와 소요 시간 관계

참 고 문 헌

[1] Franklin, S. and Graesser, A., "Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents," Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages, 1996.

[2] 최중민, "에이전트의 개요와 연구방향", 한국정보과학회 제15권 3호, pp 7-16, 1997.

[3] 박정훈, 안세용, 최중민, "계층적 작업흐름에 의한 멀티 에이전트 조정", 한국정보과학회 '98 추계학술발표논문집(2), pp 120-122, 1998.

[4] Barber, K. Suzanne and Han, David C., "Multi-Agent Planning Under Dynamic Adaptive Autonomy," IEEE International Conference on Systems, Man, and Cybernetics, 1998.

[5] E. Ephrati and J.S. Rosenschein, "Divide and Conquer in Multi-Agent Planning," In Proc. of the Twelfth National Conference on AI, pp.375-380, 1994.
[6] E. Ephrati and J.S. Rosenschein, "Multi-Agent Planning as the Process of Merging Distributed Sub-plans," The Twelfth International Workshop on Distributed Artificial Intelligence, pp.115-129, 1993.
[7] M. Georgeff, "Communication and Interaction in Multi-Agent Planning," Proc. of AAAI-83, pp125-129, 1983.
[8] 김인철, "계획 기반 에이전트 시스템", 한국정보처리학회 제4권 5호, pp 13-26, 1997.
[9] N. Carver and V. Lesser, "The Evolution of Blackboard Control Architectures," Expert Systems with Applications, Special Issue on The Blackboard Paradigm and Its Applications, vol. 7, no. 1, 1--30, 1994.
[10] FIPA98 Part 1, FIPA8711, Agent Management, Dublin, July, 1998.
[11] R.E. Fikes and N.J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," Artificial Intelligence, Vol. 2, pp.189-208, 1971.
[12] Dave E. Foulser, Ming Li and Qiang Yang, "Theory and Algorithms for Plan Merging," Artificial Intelligence Journal, Vol. 57, No. 2-3, pp.143-182. 1992.
[13] Q. Yang, "A Theory of Conflict Resolution in Planning," Artificial Intelligence Journal, Special Issue on Constraint-based Reasoning, Editors: E.C. Freuder and A.K. Mackworth, Vol. 58, No. 1-3, pp.361-392. 1992.
[14] E. Ephrati and J.S. Rosenschein, "Multi-Agent Planning as the Process of Merging Distributed Sub-plans," The Twelfth International Workshop on Distributed Artificial Intelligence, pp.115-129, 1993.
[15] J. Britanik and M. Marefat, "Hierarchical Plan Merging with Application to Process Planning," Proceedings of the 1995 International Joint Conference on Artificial Intelligence (IJCAI'95), pp. 1677-1683, 1995.
[16] Craig A. Knoblock, "Generating Parallel Execution Plans with a Partial-Order Planner," Artificial Intelligence Planning Systems: Proceedings of the Second International Conference, Chicago, IL, 1994.

박 정 훈
1997년 2월 한양대학교 전자계산학과 학사. 1999년 2월 한양대학교 전자계산학과 석사. 현재 코스넷정보통신(주) 기술연구소. 관심분야는 에이전트, 전자상거래, 인공지능, 데이타마이닝, 컴포넌트 시스템

최 중 민
1984년 서울대학교 컴퓨터공학과 졸업(학사). 1986년 서울대학교 대학원 컴퓨터공학과 졸업(석사). 1993년 State University of New York at Buffalo, Computer Science 졸업(박사). 1993년 ~ 1995년 한국전자통신연구소 인공지능연구실 선임연구원. 1995년 ~ 현재 한양대학교 전자계산학과 조교수. 관심분야는 지능형 에이전트 시스템, 인공지능, 정보검색, 데이타베이스, HCI