분산 문제 해결을 위한 개념적 모델에 관한 연구

김 은 경†

요 약

본 논문에서는 가상 공유 기억장치(VSM)를 기반으로 에이전트들간의 통신 및 협조가 이루어지는 분산 문제 해결(DPS)을 위한 개념적 모델을 제안하였다. 이 모델에서 DPS 시스템을 구성하는 모든 에이전트는 분산된 환경에 있는 모든 기억장치를 단일의 공유 기억장치로서 취급하며, 따라서 그곳에 있는 모든 데이터와 작업, 실행 결과 등을 엑세스할 수 있다. 에이전트들이 데이터와 실행의 중간 결과 및 다른 에이전트에게 요청할 작업 등을 VSM에 출력하고, 또한 VSM의 내용을 읽거나 실행함으로써 하나의 복잡한 문제를 협조적으로 해결하게 된다. 또한, 본 논문에서는 제안한 모델을 기반으로 DPS 시스템을 구축할 수 있도록 Network Linda를 이용하여 DPS 시스템 개발 환경(DPS-VSM)을 설계하였다. 또, VSM을 기반으로 하는 DPS 시스템을 시뮬레이션한 예를 제시함으로써 제안한 모델의 유용성을 보이고, DPS-VSM과 기존의 다른 DAI 프로그래밍 쉘의 특성을 비교함으로써 제안한 모델을 분석하였다.

A Study on a Conceptual Model for Distributed Problem Solving

Eun Kyung Kim†

ABSTRACT

This paper proposes a conceptual model for distributed problem solving(DPS), where cooperation and communication among agents are based on 'virtual shared memory(VSM)'. In this model, all agents in a DPS sysetm view all the memory in a distributed computer system as a single shared memory, in which data, tasks, and results can be accessed by any of the agents. Several agents cooperatively solve a complex problem, as agents write data, intermediate results of execution, and tasks requested to other agents to VSM and other agents read or execute them. Also, in order to develop DPS systems based on the proposed conceptual model, this paper designs a DPS system development environment(DPS-VSM) using Network Linda. This paper shows utility of the proposed model by presenting an example of simulating a VSM-based DPS system, and analyzes the model by comparing the features of DPS-VSM with some other DAI programming shells.

1. 서 론

분산 인공지능(Distributed Artificial Intelligence : DAI)[1, 3, 9, 12, 16, 17]은 AI에서 동시성(concurrency) 및 분산(distribution) 기술을 활용하는 것과 관련해서 출현한 연구 분야이다. AI 시스템은 대개 복잡하고 대규모이면서 계산 집약적이기 때문에 다른 시스템에 비해서 처리 속도가 느린 편이다. 지금까지 이들 대부분은 단일 시스템 상에서 단일 지식원(knowledge source)으로 개발되었으며, 각각 다른 문제 해결 기법을 기초로 하고 있으나 단 하나의 지식베이스와 한 가지 제어 메커니즘을 기반으로 구축되었다. 그

† 종신회원 : 한국기술교육대학교 정보통신공학과 조교수
논문접수 : 1995년 8월 29일, 심사완료 : 1995년 11월 8일.

러나 이처럼 단일 지식원으로 대규모의 복잡한 문제를 해결하려는 시도는 제어의 포화 문제와 지식 획득의 불완전성 및 폭발적인 매치의 문제 등을 내포하게 된다. 따라서 DAI 연구가 이러한 문제점 해결을 위한 중요한 연구 분야가 되고 있으며, 특히 분산 문제 해결(Distributed Problem Solving : DPS)[1, 3, 5, 10, 14]은 다소 불완전하지만 상호 보완적인 역할을 하는 지식을 갖는 분산된 여러 에이전트들의 협력에 의해서 하나의 큰 문제를 해결하기 위한 방법을 연구하는 DAI의 한 분야이다. DPS는 공동의 목표를 위해서 상호 통신하고 협조하는 여러개의 시스템에게 문제를 분산시킴으로써, 단일 시스템 상에서의 문제 해결의 한계점을 극복할 수 있는 새로운 해결 방법을 제시하고 있다[11, 17]. 이러한 DPS는 다음의 네가지 요인때문에 현재 AI 분야의 주요 연구 과제가 되고 있다[1].

① AI의 응용 영역 자체가 원래 분산된 것이 많다.
② 하드웨어 및 통신 기술의 발전으로 인해 대규모 분산 네트워크의 구성이 가능하다.
③ 많은 AI 응용이 성공적이기 위해서, 혹은 적어도 보다 효율적이기 위해서 분산 기술이 필요하다.
④ 협조적인 문제 해결 과정의 이해를 통해서 사람과 기계가 함께 일하는 상황을 관리하는 방법을 배울 수 있다.

본 논문에서는 하나의 복잡한 AI 문제를 분산된 여러 에이전트들의 협조에 의해서 해결하기 위해, 에이전트들이 단일의 논리적인 가상 공유 기억장치(Virtual Shared Memory : VSM)[2, 4]를 통해서 상호 통신하고 협조하도록 하는 DPS의 개념적인 모델을 제안하였다. 본 모델에서 DPS 시스템을 구성하는 각 에이전트는 분산된 시스템에 있는 모든 기억장치를 단일의 공유 기억장치로서 취급하며, 따라서 그곳에 있는 모든 데이터와 작업(task), 실행 결과 등을 엑세스할 수 있다. 에이전트들이 데이터와 실행의 중간 결과 및 다른 에이전트에게 요청할 작업 등을 VSM에 출력하고, 또한 VSM의 내용을 읽거나 실행함으로써 하나의 복잡한 문제를 협조적으로 해결하게 된다. 또한, 이 모델을 기반으로 DPS 시스템을 개발할 수 있도록 Network Linda[2, 4]를 이용하여 DPS 시스템 개발 환경을 설계하였으며, 시뮬레이션을 통해서 제안한 모델의 유용성을 보이고, 다른 DPS 모델과의 특성을 비교하였다.

2. 분산 문제 해결을 위한 개념적 모델

DPS 시스템의 구조는 해결하려는 문제 자체의 구조나 문제 해결의 기초가 되는 하드웨어적인 환경과 깊은 관계가 있으며, 무엇보다도 시스템 개발에 기본이 되는 DPS 모델이 제공하는 통신 및 협조 메커니즘의 효율성이 시스템의 성능을 좌우한다고 볼 수 있다. 따라서, 보다 효율적이고 다양한 통신 및 협조 메커니즘을 제공하는 DPS 모델에 관한 연구가 필요하며, 이를 기반으로 시스템을 개발할 수 있는 DPS 시스템 개발 환경에 관한 연구가 병행되어야 한다. 최근 DAI 분야의 연구 결과로 얻어진 아이디어가 분산 데이터 베이스의 정보 접근과 통합, 인터넷 상에서의 지능형 에이전트, 인간-컴퓨터 상호작용 등과 같은 분야에서도 그 유용성이 입증되고 있으며, DPS 구조와 다중 에이전트 학습, 상호작용 프로토콜 및 다중 에이전트 계획 등과 같은 새로운 연구 분야가 등장하였다. 특히 DAI 분야에서 주목받고 있는 몇가지 개념적인 모델이 있으며, 먼저 칠판(blackboard) 시스템 모델은 칠판이라는 대규모의 공유 기억장소와 전역적 스케줄러를 기반으로 하는 모델로서, 실행해야 할 지식원을 선택하기 위해서는 전역적 스케줄러에게 계산 부하가 집중된다. 다음, 계약 네트(contract net) 모델은 작업 할당(task allocation)이 입찰(bidding)에 의해서 행해지는 모델로, 작업의 융통성 있는 할당에 역점을 두어 지식원 간의 통신 규약을 제시하는 모델이다. 그밖에 단일 에이전트가 한 집단의 에이전트들을 위한 상세한 계획을 구축하는 모델과 메시지 전달을 기반으로 상호 통신하는 행위자(actor)들의 집단이라는 개념에 기초한 Actor 모델 등을 들 수 있다[18].

현재 대부분의 DAI 시스템 개발 환경들은 메시지 전달에 기반을 둔 통신 메커니즘을 제공하고 있다. 그러나 이러한 메시지 전달 접근 방법

은 프로그래머가 프로세스 간 통신의 매우 낮은 수준의 처리까지를 책임져야므로 시스템의 개발을 매우 어렵게 할 뿐만 아니라, 과다한 메시지 전달로 인한 통신 오버헤드는 오히려 시스템의 성능을 저하시킬 수 있다. 한편, 가상 공유 기억장치(VSM) 접근방법에서는 낮은 수준의 세부적인 사항이 시스템 개발 환경에 의해서 모두 처리되므로 시스템의 개발이 매우 용이하고, 시스템 개발자는 통신 오버헤드 문제를 염려하지 않아도 된다.

2.1 메시지 전달과 공유 기억장치 접근 방법

프로그램의 병렬 실행을 위한 두가지 주된 접근 방법으로 메시지 전달(message passing)과 공유 기억장치 모델을 들 수 있다. 이 두가지 모델은 여러가지 측면에서 차이가 있으나 특히 여러 에이전트들 사이에 공유되는 데이터를 저장하는 방법과 시스템이 실행됨에 따라 저장된 데이터를 에이전트들이 사용하는 방법에 차이가 있다.

메시지 전달 접근방법[4]에서는 각각의 데이터가 어떤 특정한 에이전트(프로세스)에 속하게 되며, 시스템이 수행되면서 필요하다면 다른 에이전트에게 명확하게 메시지를 전달해야 한다. 메시지를 송수신하기 위해서는 송신하는 에이전트와 수신하는 에이전트 모두 여러 단계를 거쳐야 하며, 통신할 에이전트를 명시하고 에이전트가 할당될 프로세서를 명시하는 등 세부적인 사항을 프로그래머가 모두 고려하여야 한다. 메시지 전달 모델을 기반으로 구축된 DPS 시스템은 통신 오버헤드로 인해서 시스템의 성능이 저하될 수 있으며, 프로세서의 수가 많아질수록 통신 복잡도는 더욱 증가한다.

공유 기억장치에 기반을 둔 통신은 에이전트들 간에 주고 받을 정보를 공유 기억장치에 쓰거나 공유 기억장치에서 읽음으로써 간단히 통신이 이루어질 수 있는 반면, 공유 기억장치가 배제된 분산된 환경에서는 부적합하다. 따라서, 본 논문에서는 분산된 환경에서 공유 기억장치에 기반을 둔 통신이 가능하도록 지원하는 가상 공유 기억장치(VSM) 접근방법[2, 4]을 채택하였다. VSM 접근방법은 에이전트들간의 통신과 협력이 단일의 논리적인 가상 공유 기억장치에 의해서 이루어지는 방법이다. 프로그램의 병렬 실행에서 가장 중요한 것은 에이전트 상호간의 통신과 협력으로 작업 사이에 데이터와 처리 결과 등을 교환할 수 있어야 하고, 어떤 작업이 필요로 하는 데이터가 있을 때 이를 이용할 수 있도록 보장해 주어야 한다. VSM 접근 방법에서는 각 작업이 다른 작업을 직접 엑세스하지 않으며, 모든 데이터의 교환이 VSM을 통해서 이루어진다. 예를 들어 한 작업이 중간 결과를 생성했을 때 그 결과를 어느 작업이 필요로 하는 지에 대해서 전혀 알 필요가 없으며, 새로 생성한 결과를 태그(tag)를 붙여서 VSM에 출력하면 되며, 이를 원하는 작업은 누구든지 가져갈 수 있다. VSM 접근 방법은 다음과 같은 장점을 갖고 있다 :

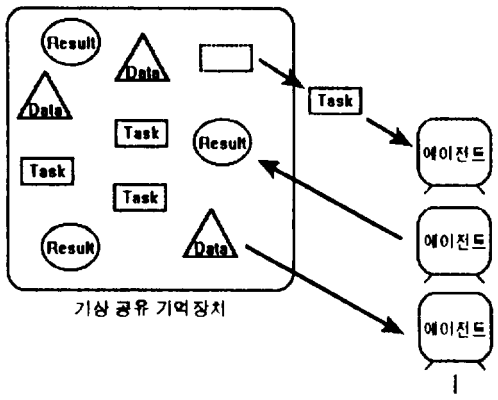
① 학습과 이용이 간단하므로 기존 프로그램의 병렬화가 용이하고 새로운 프로그램의 개발이 쉽다.
② 동적 부하 균형(dynamic load balancing)과 같은 향상된 병렬 실행의 특성이 쉽게 구현될 수 있다.
③ 컴퓨터 구조의 특수성에 따른 저수준의 세부 사항들이 사용자에게는 감추어지므로 이식가능한(portable) 프로그램의 작성이 용이하다.

VSM 모델을 기반으로 한 DPS 시스템에서 각 에이전트는 동일한 데이터 공간을 보고 있으며, 간단한 오퍼레이션을 이용해서 공유 데이터를 읽거나 쓸 수 있다. 또한, 에이전트는 다른 에이전트와 직접적으로 통신할 필요가 없으며, 따라서 통신할 에이전트를 명시할 필요가 없고, 결과적으로 위치 및 접근 투명성(transparency)이 제공된다. VSM은 칠판과 같이 실제로 공유되는 기억장치가 아니라 가상적으로 공유되므로 비록 기억장치가 네트웨으로 연결된 여러 프로세서에 물리적으로 분산되어 있다해도 프로그래머는 이를 하나의 거대한 기억장소 풀(pool)로 생각할 수 있다.

2.2 에이전트 구조

본 논문에서 제안한 DPS 모델의 각 에이전트는 전체 문제의 일부분을 해결하는데 필요한 지

식을 포함하면서 병렬로 실행되며, 각각이 독립된 추론 엔진(Inference Engine)과 지식 베이스(Knowledge Base : KB) 및 VSM 관리기(VSM Manager : VSMM)로 구성된다. 각 에이전트의 추론 엔진(Inference Engine)은 에이전트가 자신의 전문 분야와 관련된 문제를 해결할 수 있도록 지원한다. 추론 엔진의 지역적인 실행 싸이클(local execution cycle)은 자신의 문제 해결 동안 생성된 내부적인 처리에 의해서 개시되거나, VSMM에 의해서 전달된 외부적인 사건에 의해 개시된다. 또한, 각 에이전트는 문제 해결을 위해서 지역적으로 유지되는 객체 지향 KB를 포함하며, 전체 문제의 한부분을 해결할 수 있는 영역 전문 지식과 다른 에이전트와 협조하기 위해서 필요한 메타 지식(meta knowledge)을 포함한다. 한편, 영역 전문 지식과 메타 지식은 모두 규칙 형태로 표현한다. 각 에이전트는 VSM을 통해서 통신이 이루어지며, VSMM은 에이전트가 VSM의 내용을 읽고 쓸 수 있도록 지원함으로써 에이전트 간의 정보 교환을 지원하고, 다른 에이전트로부터의 요구에 응답한다. 이를 위해서 VSMM은 탐색기(searcher)와 제동기(trigger)라는 2개의 모듈로 구성하였다. 탐색기는 지역적인 KB에 없는 정보를 VSM에서 찾거나 공유될 필요가 있는 정보를 VSM에 추가하며, 제동기는 다른 에이전트의 요구에 따라 에이전트의 실행 싸이클을 개시한다.

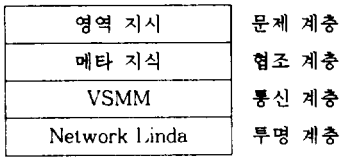


(그림 1) 분산 문제 해결 모델(제안)
(Fig. 1) DPS model(proposed)

2.3 분산 문제 해결(DPS) 모델

본 논문에서 제안한 DPS 모델을 도식화하면 (그림 1)과 같다.

이 모델에서 DPS 시스템을 구성하는 각 에이전트는 분산된 시스템에 있는 모든 기억장치를 단일의 공유 기억장치로서 취급하며, 따라서 그곳에 있는 모든 데이터와 작업, 실행 결과 등을 엑세스할 수 있다. 에이전트들이 데이터와 실행의 중간 결과 및 다른 에이전트에게 요청할 작업 등을 VSM에 출력하고, 또한 VSM의 내용을 읽거나 실행함으로써 하나의 복잡한 문제를 협조적으로 해결한다. 이 모델을 계층 구조로 표현하면 (그림 2)와 같다.



(그림 2) 분산 문제 해결 모델의 계층 구조
(Fig. 2) Hierarchical structure of the DPS model

(1) 투명 계층(transparency layer)

생성된 에이전트를 프로세서에 할당하고 실제로 VSM을 관리하는 기능은 이 계층에서 제공하며, 이러한 기능은 뒤에서 기술할 Network Linda가 지원한다. 즉, 프로세스간 통신과 부하 균형(load balancing), VSM 관리 등의 모든 저수준 처리는 이 계층에 의해서 사용자에게 감추어진다.

(2) 통신 계층(communication layer)

이 계층에서는 에이전트가 VSM을 통해서 통신할 수 있도록 VSMM이 여러가지 통신 기능을 지원한다. 즉, VSMM은 VSM에 있는 공유 데이터를 KB에 추가시킴으로써 에이전트들 간의 통신을 지원하고, 인터프리터가 메타 지식에 따라 적절한 행동을 수행하도록 지원하며, 또한 KB에 있는 데이터를 VSM에 추가시키는 기능을 수행한다. 또한, 한 에이전트가 병렬로 실행될 수 있는 다른 에이전트를 생성할 필요가 있을 때 VSMM에게 요청할 수 있으며, 이를 모두 이 계층에서 처리한다.

(3) 협조 계층(cooperation layer)

이 계층에서는 문산된 에이전트들 간의 효율적인 협조를 위해서, 해결하고자 하는 문제의 특성에 적합한 협조 메커니즘을 기본으로 하는 각종 서비스를 제공하게 되며, 메타 지식으로 표현한다. 메타 지식은 영역 지식과 마찬가지로 생성 규칙으로 표현할 수 있다.

(4) 문제 계층(problem layer)

이 계층에는 각 에이전트의 문제 해결에 필요한 영역 지식을 표현하며, 영역 지식의 표현 방법은 3.2절에 기술하였다.

3. 분산 문제 해결 시스템 개발 환경

본 논문에서는 제안한 DPS 모델을 기반으로 시스템을 개발할 수 있도록 DPS 시스템 개발 환경을 설계하였다. 이 개발 환경은 LAN으로 연결된 워크스테이션 상에서 Network Linda[2, 4]라는 병렬 계산 환경을 기초로 하고 있다.

3.1 Network Linda

Scientific사의 Linda[2, 4]는 LAN과 공유 및 분산 기억장치 다중프로세서를 위한 병렬 계산 환경을 제공하며, Network Linda는 Linda의 유닉스 네트웨 버전이다. Network Linda는 동질(homogeneous) 및 이질적인(heterogeneous) 네트웍 상에서 병렬 응용이 실행될 수 있도록 지원한다. 또한, 병렬 프로그램을 구성하는 모든 프로세스에게 단일의 논리적으로 공유된 기억 장치를 제공함으로써 이를 통해서 통신 및 협조가 가능하도록 지원하는 시스템이다. 모든 프로세서가 기억장치에 대해서 동일한 관점을 가지며, 모든 데이터에 동일하게 접근할 수 있다는 의미에서 Linda의 기억장치는 대칭적(symmetric)이다. Linda에서 각 프로세스는 동일한 데이터 공간을 보고 있으며, 간단한 오퍼레이션을 이용해서 공유 데이터를 읽고 쓸 수 있다.

Linda와 같은 VSM 기반 시스템은 메시지 전달 라이브러리와는 달리 병렬 프로그래밍을 위한 고수준의 조정(coordination) 언어를 포함하고 있으며, 이를 이용하여 기존의 순차적인 소프트웨어를 병렬 소프트웨어로 변경할 수 있다. Linda는 튜플 공간(tuple space : TS)이라는 기억장치 안에서 병렬성을 구현한다. TS는 튜플이라는 일련의 데이터들을 모아놓은 것으로, 모든 프로세스에 의해서 공유되며, 튜플은 타입(정수형, 실수형, 문자열 등)이 구분되는 일련의 필드들로 구성된다. 예를 들면, ("sample tuple", 2, 12.3)은 각각 문자열, 정수형, 실수형인 3개의 필드로 구성된 튜플이다. 템플릿(template)은 형식적(formal) 필드가 허용되는 것을 제외하면 튜플과 동일하며, 형식적 필드는 튜플 필드의 값을 지역 변수에 할당하기 위해서 사용된다. 템플릿과 튜플은 모든 필드가 같은 타입, 같은 길이, 또는 같은 상수(literal) 값을 가질 때 매치된다. 예를 들면, 템플릿("sample tuple", i, ?x)는 변수 i와 x가 각각 정수형과 실수형으로 정의되어 있을 때 튜플("sample tuple", 2, 12.3)과 매치될 수 있다. ?x는 형식적 필드를 나타내며, x에 12.3이 할당된다. TS는 네트웨 상의 노드들에 분산되어 있는 전역적으로 공유된 연상 기억장치(associative memory)이다. 즉, 튜플들은 주소가 없으며, 튜플을 구성하는 필드의 내용에 의해서 참조된다. Linda는 다음과 같은 4가지의 기본 연산을 제공한다[4] :

① out(t)－튜플 t를 TS에 추가한다(write).
② in(s)－TS에서 템플릿 s와 매치하는 튜플을 읽은 후 제거한다(extract).
③ rd(s)－TS에서 템플릿 s와 매치하는 튜플을 읽기만 한다(copy).
④ eval(t)－튜플 t를 평가(evaluation)하기 위해서 동시 진행적인 새로운 프로세스를 생성한다.

'out' 연산은 튜플을 TS에 추가한다. 만약 필드 가운데 하나가 연산식을 포함하면, TS에 추가하기 전에 먼저 연산식을 평가한다. 'in' 연산은 인자로 기술된 템플릿과 매치하는 튜플을 TS에서 제거하기 위해서 먼저 탐색을 시도한다. 만약 매치하는 튜플이 있으면 TS에서 제거하고, 템플릿에 형식적 필드가 있으면 대응하는 튜플의 필드 값이 할당된다. 만약 매치하는 튜플이 없으면, 매치하는 것이 있을 때까지 기다린다. 'rd' 연산은 매치된 튜플을 TS에서 제거하지않는 것을 제외하면 'in' 연산과 동일하다. 즉, 단지 TS

의 내용을 읽기만 한다. 'eval' 연산은 각 필드를 평가하기 위해서 새로운 프로세스를 생성한다.

위의 4가지 연산은 프로그램이 여러 프로세스를 생성하거나 네트웨을 통해서 프로세서들로부터 처리 결과의 수집 및 데이터 저장, 프로세서 간의 통신 관리 등과 같은 병렬 처리에 필요한 여러 오퍼레이션을 수행할 수 있도록 한다. Linda는 사용자가 분산된 컴퓨터 시스템에 있는 모든 기억장치를 단일 공유 기억 장치처럼 볼 수 있도록 하며, 따라서 한 기억장치에 있는 데이터나 처리 결과 등을 어떤 프로세스라도 엑세스할 수 있다.

3.2 에이전트 구문

한편, 객체(object)는 단일화된 통신 프로토콜이 제공되는 독립된 개체를 의미하며, 따라서 하나의 시스템을 여러개의 객체로 나누는 것은 시스템의 구조를 매우 자연스럽게 한다. 특히 단일 통신 프로토콜과 독립성(self-containedness)으로 인해 외부로부터의 불법적인 접근을 방지하게 되므로 객체 간의 상호작용이 명확해지며, 이러한 장점은 병행성 표현을 위한 거의 완벽한 기초를 제공하게 된다[16, 19]. 따라서, 본 논문에서는 병렬 및 분산 처리의 기본 단위로 취급할 수 있는 각 에이전트를 객체에 대응시킴으로써 에이전트 자체의 구조를 보다 명확히 하였다. 또한 각 에이전트의 지식은 OPS5와 유사한 생성 규칙 형태로 표현하였다. (그림 3)은 에이전트 구문(syntax)의 일부분을 BNF(Backus-Naur Form) 정의로 표현한 것이다. 한편, 에이전트 정의시에 동일한 프로그램 코드를 중복해서 정의하지않기 위해서 에이전트 간에 지식의 공유가 필요하며, 이를 위해서 본 논문에서는 다중 상속(multiple inheritance) 메커니즘을 도입하였다. 그러나 상속 개념은 원래 객체 지향 프로그래밍에서 빌려온 것으로, '지식 표현'이라는 점에서 고려되지 않았다. 따라서 본 논문에서는 상속 메커니즘을 지식 표현 기법인 '규칙'의 특성을 반영하도록 변형하였다. 에이전트의 행위는 하나 이상의 규칙으로 표현되며, 경우에 따라서는 다른 에이전트의 행위를 나타내는 규칙의 일부만을 포함하거나(subset 관계), 또는 일부와는 동일하고 일부와는 다를 수 있다. 다시 말하면, 규칙이라는 지식 표현에서의 공유 메커니즘은 부분적이거나 선택적인 공유를 허용하는 것이 바람직하다. 따라서

```
<agent_def> : : = <agent_spec>
  {<agent_meta>}{<agent_body>}
<agent_spec> : : =
  '("agent'<agent_name><slots>')'
<slots>  : =
  {'comment : '{<word>}+}
  {'ancestor : '{<agent_name>{<type>}+}+}
  {'element : '{<wme_def>}+}
  {'initial : '{<init_clause>}+}
<agent_body> : : =
  '("domain'<agent_name>{<production>}+')'
<agent_meta> : : =
  '("meta'<agent_name>{<production>}+')'
<agent_name> : : = <identifier>
<identifier> : : = <alphabet>{'_' | <digit>}*
  {<identifier>}*
<alphabet> : : = ['a'_'z'] | ['A'_'Z']
<word> : : = <identifier> | {<digit>}+ |
<special_char>{<word>}*
<digit> : : = ['0'_'9']
<special_char> : : = '.' | ',' | ';' | ' : ' | '?' | '!'
<attri_value> : : = <attri><value>
<attri> : : = '-'<identifier>
<value> : : = <alphabet> | <digit> | <identifier>
   | <relation_stmt> | <or_stmt> | <function_stmt>
<wme_def> : : = <wme_name> {<attri>}*
<wme_name> : : = <identifier>
<production> : : = '("rule'<rule_name>
                    <cond_stmt>'→'<act_stmt>')'
<relation_stmt> : : = '{'<arg1><relop><arg2>'}'
<relop> : : = '>' | '<' | '=' | '=<' | '=>' | '><'
<arg1> : : = <digit> | <var> | <function_stmt>
<or_stmt> : : = '<<'<arg2>{<arg2>}+'>>'
<arg2> : : = <digit> | <alphabet>
<function_stmt> : : = '('<binop><arg1><arg1>')'
<binop> : : = '+' | '-' | '*' | '/' | 'mod'
<type> : : = ' : equal' | ' : add'{<rule_name>}+
   | ' : delete'{<rule_name>}+
   | ' : replace'{<rule_name>'='>'<rule_name>}+
<rule_name> : : = <identifier>
<init_clause> : : = {<wme_creation_stmt>}*
   | {<agent_creation_stmt>}*
<wme_creation_stmt> : : = '("make'<wme>')'
<wme> : : = <wme_name>{<attri_value>}*
<agent_creation_stmt>
   : : = '("create'<agent_name>')'
<cond_stmt> : : = {{<wme_cond>}*{<vsm_cond>}*}+
<wme_cond> : : = '('<wme>')'
   | '{"('<wme>')'<var>'}'
   | '-"('<wme>')'
<act_stmt> : : = {{<wme_action_stmt>}*
  {<vsm_action_stmt>}*}+
<wme_action_stmt> : : = '("make'<wme>')'
   | '("remove'<var>')'
   | '("modify'<var>{<attri_value>}+')'
   | '("bind'<var>'("accpet")")'
   | '("write'{<format_fn>}<list>')'
   | '("halt")'
<vsm_cond> : : = <template>
<template> : : = '("""'<template_name>'""
{','<variable> | ','<formal>}*')'
```

(그림 3) 에이전트의 구문
(Fig. 3) Syntax of an agent

본 논문에서는 상속의 형태를 구분하여 지정할 수 있도록 상속 메커니즘을 확장하여 'ancestor' 슬랏에 공유의 형태를 기술할 수 있도록 하였다. 'ADD' 형태는 상속 개념의 '구체화'와 유사하며, 'EQUAL'은 행위 부분의 규칙들은 동일하나 나머지 슬랏의 내용이 다른 경우에 유용하다. 원래의 상속 개념은 'one-instance one-class' 라는 문제점을 내포한다. 즉, 에이전트의 행위를 나타내는 규칙 가운데 단 하나가 다르거나 불필요하더라도 새로운 클래스를 정의해야 한다. 그러나 이러한 문제점은 상속시 공유의 형태를 명시함으로써 해결 가능하다[13].

3.3 통신 프리미티브

Network Linda는 앞에서 설명한 네가지 연산을 기본으로 하는 조정 언어를 제공하고 있으며, 이를 이용하여 본 개발 환경의 통신 프리미티브를 구현할 수 있다. 본 논문에서 제안한 DPS 모델에서는 에이전트들이 VSM을 이용하여 상호 통신하므로 VSM에 튜플, 즉 공유 작업메모리요소(working memory element)를 입출력시키는 다음의 우변 행위(right-hand side action)를 이용하여 통신이 가능하다 :

① 공유 데이터 출력 행위(vsm-out) : VSM에 데이터나 처리 결과를 출력시킨다.
예) (vsm-out (sample ^x 10 ^y 20)) – VSM에 튜플 'sample'을 출력한다.

② 공유 데이터 복사 행위(vsm-read) : VSM에 있는 데이터나 처리 결과를 자신의 KB로 읽어들인다.
예) (vsm-read (sample ^x <a> ^y <b>)) – 만약 튜플(sample ^x 10 ^y 20)과 매치되면 KB로 복사하고, 변수 a와 b에는 각각 '10'과 '20'이 할당된다.

③ 공유 데이터 추출 행위(vsm−in) : VSM에 있는 데이터나 처리 결과를 자신의 KB로 읽어들인 후 VSM에서 제거시킨다.
예) (vsm-in (sample ^x 10 ^y <a>)) – 튜플(sample ^x 10 ^y 20)과 매치되어 KB로 이동된 후 VSM에서는 제거된다.

4. 분산 문제 해결(DPS) 시스템의 구축 예

본 논문에서는 제안한 개념적 모델의 유용성을 보이기 위해서 신장질환 진단 전문가 시스템 RDES(Renal-disease Diagnosis Expert System) [22]의 1차 진단 모듈을 DPS 시스템으로 구축하였다. 즉, 1차 진단 모듈을 환자의 병력과 소변 중상, 기타 중상, 발병 유형 및 신체 검사 결과를 기반으로 진단하는 다섯개의 분산된 에이전트로 구성하고, 이들간의 통신 및 협조에 의해서 신장 질환을 진단하도록 DPS 시스템을 구축하였다. RDES는 Nexpert Object[20]와 OI(Open Interface)[21]를 이용하여 개발하였으며, 이에 대한 자세한 설명은 참고문헌 [22]로 대신하겠다. DPS 시스템의 각 에이전트를 자신의 지역적인 KB와 추론 엔진 및 VSMM으로 구성하고, 이들간의 통신은 VSM을 통해서 이루어지도록 하였다. 따라서 각 에이전트를 별도의 KB로 구축하고, 각 에이전트의 VSMM은 외부 프로시쥬어(external procedure)로 작성하여 에이전트들

(1) [ACTIVE]	계산 자원이 할당되어 인식-행위 싸이클을 실행하는 상태
(2) [END]	정상적으로 종료한 상태
(3) [REQUEST]	VSMM에게 공유 데이터의 read/write를 요청한 상태
(4) [WAIT]	계산 자원의 할당을 기다리는 상태

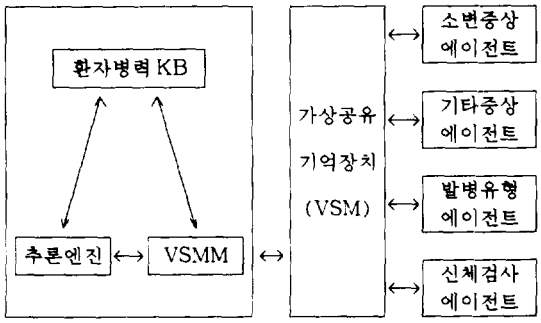
(그림 4) 에이전트의 상태 구분
(Fig. 4) States of an agent

[1] 최상위 레벨에서 생성한 순서대로 각 에이전트를 큐에 삽입한다.
[2] 큐에서 [END] 상태가 아닌 한 에이전트를 삭제하여 그 상태를 [ACTIVE]로 수정하고, 에이전트의 인식-행위 싸이클을 다섯번 반복한 후 에이전트의 상태를 [WAIT]로 바꾸고 종료한다. 단, 다음 중 하나에 해당하면 도중에 종료한다.
① VSMM을 호출하면 상태를 [REQUEST]로 바꾸고, 제어를 VSMM에게 넘긴 다음 에이전트를 큐에 삽입한 후 종료한다.
② 더 이상 매치하는 규칙이 없으면 상태를 [END]로 바꾼 다음 종료한다.
[3] 모든 에이전트의 상태가 [END]가 될 때까지 단계 [2]를 반복한다.

(그림 5) 전처리기의 실행 알고리듬
(Fig. 5) Execution algorithm of the preprocessor

간의 통신이 VSMM의 제어에 의해서 VSM을 통해서만 가능하도록 하였다. 또한, 이들 다섯개의 에이전트를 순차적인 실행 환경에서 의사 병렬(quasi-parallel)로 실행하기 위해서 각 에이전트의 상태를 (그림 4)와 같이 구분하였으며, 각 에이전트에게 계산 자원이 할당될 때마다 그 인식-행위 싸이클을 다섯번씩(본 시스템의 규칙의 최대 조건수가 다섯개이므로 임의로 선택한 반복 횟수임) 실행하도록 전처리기(preprocessor)를 설계하였다. (그림 5)는 프리프로세서의 실행 알고리듬이며, 본 DPS 시스템을 도식화하면 (그림 6)과 같다.

각 에이전트가 자신만의 진단 지식을 기반으로 진단하는 도중에 다른 에이전트의 지식이 필요하면, 규칙의 우변에서 Nexpert Object의 'execute' 오퍼레이터를 사용하여 자신의 VSMM을 환자병력 에이전트



(그림 6) VSM 기반 DPS 시스템의 구조
(Fig. 6) A VSM-based DPS system architecture

```
(@RULE=UTI1
  (@LHS=
    (= (urine.pain) ("yes"))
    (= (urine.volume) ("normal"))
    (= (urine.pattern) ("urgency"))
    (= (urine.color) ("cloudy"))
  )
  (@HYPO=UTI 01)
  (@RHS=
    (assign (2) (UTI.score))
    (execute ("vsmm1") (@ATOMID=UTI.score;))
  )
)
```

(그림 7) VSMM 호출의 예
(Fig. 7) An example of VSMM calling

호출한다. 본 시스템에서는 이러한 공유 데이터를 OI에서 제공하는 Script 언어에서 전역 변수로 선언하여 관리함으로써 지역 데이터와 구분하였다. 즉, 전역 변수들을 VSM에 있는 공유 데이타로 취급하였다. (그림 7)은 소변증상 에이전트가 UTI.score라는 값을 VSM에 출력하기 위해서 VSMM을 호출한 예이다.

신장질환 진단은 문제 자체가 갖는 분산적인 성격이 풍부하지 않으므로 본 논문에서 제안한 개념적 모델의 장점을 충분히 발휘할 수 없었으며, 시스템 개발에 사용한 전문가 시스템 개발 도구인 Nexpert Object 역시 원래 순차적인 처리만을 지원하므로 제안한 모델을 시뮬레이션하는데 어려움이 있었다. 그러나 이러한 시뮬레이션을 통해서 제안한 모델의 유용성을 확인할 수 있었으며, 실제로 분산된 계산 환경에서 구축하지 않았으므로 처리 속도 등의 성능 향상은 비교하지 않았다. 한편, 전체 시스템을 분산 가능한 여러개의 에이전트로 분해함으로써 KB의 복잡도가 감소하고, 새로운 지식의 획득에 따른 KB의 유지 보수가 훨씬 용이하게 되었다. 특히 실제로 분산된 환경에서 각 에이전트가 모든 기억장치를 하나의 거대한 공유 기억장치로 취급함으로써, 자신이 필요한 지식이 어느 에이전트에 속하는지를 모르고도 지식의 공유가 가능하므로 다른 에이전트와의 통신 및 협조가 매우 용이하다. 따라서, 문제 자체가 기능적으로 분산된 에이전트들로 구성된 응용 분야에 보다 적합할 것으로 판단한다.

5. 비교 분석

기존의 대표적인 DAI 프로그래밍 쉘과 본 개발 환경의 특징을 비교하면 〈표 1〉과 같다. 〈표 1〉의 대부분의 쉘이 생성 규칙으로 지식을 표현하며, 본 개발 환경에서는 각 에이전트를 객체 개념에 대응시킴으로써 에이전트 자체의 구조를 보다 명확히 하였다. COPS[14]의 경우 사용자가 칠판의 역할을 할 프로세스를 지정해야 하며, 칠판으로 지정된 프로세스의 작업 메모리에 있는 데이터를 다른 프로세스가 원격으로 읽고 쓸(remote reading/writing) 수 있도록 하였다. 그러

나 본 개발 환경에서는 모든 에이전트가 계산 서버(computational server) 및 VSM 서버로서의 역할을 할 수 있으므로, VSM을 관리하기 위해서 프로그래머가 특정 에이전트를 지정할 필요가 없다. TB(Transactional Black-boards)[6]의 칠판은 lisp의 sexpression 형태의 데이터를 저장하는 저장소로서, 칠판에 저장된 데이터에 대한 접근은 거래(transaction) 안에서만 허용되고 거래 관리자에 의해서 관리된다. 또한, TB에서는 에이전트들의 초기 분산 형태가 시스템 설계자에 의해서 명시되어야 하므로 동적인 부하 균형이 이루어질 수 없는 반면, 본 개발 환경의 경우 동적인 부하 균형이 가능하다. 그리고, 본 개발 환경의 경우 데이터 유도(data-driven) 방식으로 처리된다. 이는 VSM에 있는 값을 엑세스하는 것이 자동으로 에이전트의 행위를 개시(trigger) 시킴을 의미한다. 또, TB의 흑판은 물리적으로 공유된 단일 기억장소이지만, 본 개발 환경의 공유 기억장치는 가상적으로 공유되므로 비록 물리적으로 분산되어 있다해도 프로그래머는 이를 하나의 거대한 기억장소로 생각할 수 있다. DAIS [12]는 접근 투명성만을 제공하는 반면, 본 개발 환경은 접근 투명성과 위치 투명성을 모두 제공한다. GRATE[11]의 에이전트는 두개의 다른 요소, 즉 협조 및 제어 계층과 영역 레벨 시스템으로 구성된다. 영역 레벨 시스템은 미리 존재하거나 목적에 맞게 구축되는 문제 해결 부분이다. 협조 및 제어 계층은 메타 제어기(meta controller)로서, 본 개발 환경의 메타 규칙과 유사한 역할을 한다. GRATE 에이전트는 다른 에이전트를 표현하는 이웃 모델(acquaintance model)과 지역적인 영역 레벨 시스템의 추상적인 관점을 표현하는 자기 모델(self model)로 구성된다. 따라서 각 에이전트는 다른 에이전트들에 대해서 미리 알고 있어야 한다는 제한점이 있다. MACE [7] 에이전트는 메시지 전달을 통해서 통신하는 능동적인 객체이며, 각 에이전트는 자기 자신과 다른 에이전트들의 역할과 기능, 주소 등으로 구성된 모델을 갖고 있다. 특히 주소는 에이전트의 위치를 나타내며, 에이전트 이름, 클래스, 위치한 노드 및 에이전트가 실행되고 있는 기계로 구성되고, 다른 에이전트와 통신할 때 이 주소로 에이전트를 지정하게 된다. 한편, MACE와 AF (Activation Framework) [8]를 제외한 대부분의 지식원은 속성(attribute)에 의해서 참조되며, 이는 다른 에이전트의 주소를 알아야만 통신할 수 있는 형태보다 훨씬 융통성이 있다. AF의 AFO(AF Object)는 통신할 모든 다른 에이전트가 알고 있는 전역적인 이름을 가지며, 메시지는 이름에 의해서 AFO에 전달된다. DVMT[15]의 각 에이전트는 다른 에이전트와 통신할 수 있는 반독립적인 문제 해결 노드이며, DVMT는 기능적으로 정확하고 협조적인(functionally-accurate, ooperative : FA/C) 접근방법에 기초한 협조 메커니즘을 제공하고 있다. 한편, 본 개발 환경에서는 자체적으로 제공하는 협조 메커니즘은 없으며, 문제의 성격에 적합한 협조 메커니즘을 메타 규칙으로 표현해야 한다.

〈표 1〉 분산 인공 지능 프로그래밍 쉘의 특성 비교
〈Table 1〉 Comparision of different DAI shells

개발 환경 / 특성	DAIS	COPS	GRATE	TB	DVMT	MACE	AF	DPS-VSM (제안)
공유 기억장치		BB		BB	BB			VSM
에이전트 표현	DAIS /KR		에이전트모델		노드	객체	객체	객체
지식표현	규칙+프레임	규칙 (OPS5)	규칙		규칙	속성 집합	규칙	규칙 객체
상속성					다중	다중		다중
지식원 참조	속성	속성	속성	속성	속성	주소	이름	속성
분산단위	에이전트	프로세스	에이전트	에이전트	지식원	에이전트	AFO	에이전트
통신 메커니즘	메시지	BB, 메시지	메시지	BB, 메시지	BB, 메시지	메시지	메시지	VSM
협조 메커니즘	메시지	메시지	협조모듈		FA/C	메시지	메시지	메타 규칙

(BB : 블랙보드)

6. 결 론

메시지 전달을 기반으로 하는 분산 문제 해결(DPS) 시스템은 잦은 메시지 전달로 인한 통신 오버헤드 때문에 시스템의 성능이 저하될 수 있으며, 메시지가 특정한 에이전트에 의해서만 접근 가능할 뿐만 아니라 부하 균등의 의무가 프로그래머

에게 주어지므로 시스템의 개발이 매우 어렵다. 또한, 칠판과 같은 물리적으로 공유된 기억장소를 기반으로 하는 DPS 시스템은 실제로 분산된 환경에는 부적합하다. 본 논문에서는 DPS를 위한 개념적 모델로서 시스템을 구성하는 각 에이전트들이 단일의 가상 공유 기억장치(VSM)를 통해서 상호 통신하고 협조하는 모델을 제안하였다. VSM 기반 시스템에서는 통신 오버헤드를 염려할 필요가 없으며, VSM에 있는 모든 정보를 모든 에이전트가 동일하게 공유할 수 있다. 또한, 부하 균등과 같은 문제 세부적인 사항이 투명 계층에서 처리되므로 개발자는 시스템 자체의 개발에만 전념할 수 있으며, 비록 기억장치가 네트웨상에 있는 여러 프로세서에 물리적으로 분산되어 있다해도 개발자가 이를 하나의 거대한 기억장소로 생각하고 프로그램을 개발할 수 있다. 또한, 본 논문에서는 Network Linda를 이용하여 제안한 모델을 기반으로 DPS 시스템 개발 환경을 설계하였다. 또한, VSM을 기반으로 하는 DPS 시스템을 시뮬레이션한 예를 제시함으로써 제안한 모델의 유용성을 보이고, DPS-VSM과 기존의 다른 DAI 프로그래밍 쉘의 특성을 비교함으로써 제안한 모델을 분석하였다.

앞으로 본 논문에서 설계한 DPS 시스템 개발 환경을 Ethernet으로 연결된 워크스테이션 상에서 C와 Network Linda를 이용하여 구현하고, 개발 환경 자체가 보다 다양한 협조 메커니즘을 직접적으로 지원하는 방법에 대해서 연구할 계획이다.

참 고 문 헌

[1] Bond, A.H. and Gasser, L.(ed.), Readings in Distributed Artificial Intelligence, Morgan Kaufmann, pp.2-35, 1988.

[2] Cagon, L. and Sherman, A.H., "Linda units network systems," IEEE Spectrum, Vol.30, No.12, pp.31-35, 1993.

[3] Cammarata, S., McArthur, D., and Steeb, R., "Strategies of Cooperation in Distributed Problem Sloving," IJCAI'83, pp.767-770, 1983.

[4] Carricro, N.J., Gelernter, D., et. al., "The Linda alternative to message-assing systems," Parallel Computing 20, pp.633-655, 1994.

[5] Durfee, E.H., Coordination of Distributed Problem Solvers, Kluwer Academic Pub., pp.1-44, 1988.

[6] Ensor, J.R. and Gabbe, J.D., "Transactional Blackboards," in Readings in Distributed Artificial Intelligence, Bond, A. H. and Gasser, L.(ed.), Morgan Kaufmann, pp.557-561, 1988.

[7] Gasser, L., Braganza, C., and Herman, N., "MACE : A Flexible Testbed for Distributed AI Research," in Distributed Artificial Intelligence, M.N. Huhns, ed., Morgan Kaufmann, pp.119-152, 1987.

[8] Green, P.E., "AF : A Framework for Real-Time Distributed Cooperative Problem Solving," in Distributed Artificial Intelligence, M.N. Huhns, ed., Morgan Kaufmann, pp.153-176, 1987.

[9] Han, C.H. and Parameswaran, N., "MAPS : A Multiagent Production System for Problem Solving," PRIC AI '92, pp.148-152, 1992.

[10] Hender, J., et. al., "Multiple Approaches to Multiple Agent Problem Solving," IJCAI-91, pp.553-554, 1991.

[11] Jennings, N.R., Mamdani, E. H., et.al., "GRATE : a general framework for cooperative problem sloving," Intelligent System Engineering, pp.102-114, 1992.

[12] Kannan, R. and Dodrill, W.H., "DAIS : A Distributed AI Programming Shell," IEEE EXPERT, Vol.5, No.6, pp.34-42, 1990.

[13] Kim, E.G., "DPS-Linda : A Distributed Problem Solving System Development Environment," Proc. of 3rd Int'l Conf. on Decision Support System, Vol.1, pp. 187-192, 1995.

[14] Leao, L.V. and Talukdar, S.N., "COPS : A System for Consulting Multiple Black-

boards," in Readings in Distributed Artificial Intelligence, Bond, A.H. and Gasser, L.(ed.), Morgan Kaufmann, pp. 547-556., 1988.

[15] Lesser, V.R. and Corkill, D.D., "The Distributed Vehicle Monitoring Testbed : A Tool for Investigating Distributed Problem Solving Networks," AI Magazine, Vol.4, No.3, pp.15-33, 1983.

[16] Lieberman, H., "Using Prototypical Objects to implement Shared Behavior in Object-Oriented Systems," Proceeding of OOPSLA'86, 1986, pp.214-223. 1986.

[17] Poggi, A., "DAISY : an Object-Oriented System for DIstributed Artificial Intelligence," Intelligent Agents, Wooldridge, M.J.(Eds), Springer-Verlag, pp.341-354, 1995.

[18] Sycara, K., "The present and Future of DAI : a Study in Enterprise Integration," Artificial Intelligence, Zhang, C., Debenham, J., and Lukose(ed.), World Scientific, pp.1-12, 1994.

[19] Yonezawa, A. and Tokoro M., "Object-Oriented Concurrent Programming : An Introduction", in Object-Oriented Concurrent Programming(edited by Yonezawa, A. and Tokoro M.), The MIT Press, pp. 1-7, 1987.

[20] Neuron Data, Nexpert Object Reference Manual, 1991.

[21] Neuron Data, Open Interface Reference Manual, 1991.

[22] 김은경, 김대진, 설순욱, "신장질환 진단 전문가 시스템의 설계 및 구현," 한국전문가시스템학회 추계학술대회, pp.255-262, 1995.

김 은 경

1984년 숙명여자대학교 물리학과 졸업(이학사)
1987년 중앙대학교 대학원 전자계산학과 졸업(이학석사)
1991년 중앙대학교 대학원 전자계산학과 졸업(공학박사)
1984년~86년 한국방송통신대학교 전자계산학과 조교
1992년~현재 한국기술교육대학교 정보통신공학과 조교수
관심분야 : 분산인공지능, 분산문제해결, 지능형 학습시스템 전문가시스템등