멀티 에이전트 설계 시스템(PART II)

MADS 개발에서의 문제점

MADS 어플리케이션을 개발하는 것이 매우 어려운 일이 될 수 있는데 그것은 지식 베이스의 설계가 복잡한 작업이고 또한 MADS 개발이 아직도 새로운 분야라서 그것을 시작할만한 경험이 축적되거나 소프트웨어의 도움을 받기 어렵기 때문이다. 그러나 다행스럽게도 다음과 같은 희소식도 있다. 첫째로 MADS가 현재 개발되고 있기 때문에 축적된 경험과 유용한 소프트웨어의 집합이 점점 커지고 있다는 것이다.[2,3,10] 둘째로 프로젝트의 목표가 상당히 소극적이라 할지라도 완성된 어플리케이션은 현재 가능한 것보다는 훨씬 더 나은 것이 될 가능성이 높기 때문이다. 예를 들어서, 이종기기간에 레가시(legacy) 도구의 상호운영(interoperability)을 가능하게 하거나 그것을 쉽게 바꿀 수 있게 하는 것이 합리적인 목표가 될 것이다. 이러한 것이 MADS 개발에서 발생되는 보다 복잡한 문제를 제기하지는 않지만 상호운영(interoperability)이란 이 도구의 집합 사이에서 정보의 흐름을 자동화하는 것을 말한다. 이것은 만약 그 이전에 이 정보가 수작업에 의해 취급되었다면 엄청난 이득이 되는 것이다.

MADS 어플리케이션을 만드는데 있어서 개발자는 에이전트 집합과 에이전트가 어떻게 상호작용하는 가에 대한 모델 둘 다를 설계해야 한다. 어떤 에이전트가 포함되어야 할지를 결정하고, 상호작용 모델을 확립하는 것에 대한 다양한 선택 대상이 있을 수 있다. 이 선택 대상은 현존하는 소프트웨어나 하드웨어의 요구조건이나 혹은 특정한 연구나 또는 개발 계획 등과 같은 과거의 결정에 의하여 종종 제한받는다.

⊙ MADS 에이전트 특성

협력 시스템에서의 에이전트 소스에는 요약된 레가시나 상업용 도구, 새롭게 설계된 주문 소프트웨어, 그리고 사람 등이 포함된다.

다음에 나오는 분류는 에이전트의 구체적인 특성에 관심을 집중시킨 것이다. 이러한 특성은 에이전트나 혹은 그것의 인터페이스가 긴밀하고도 효율적인 협력이 가능하게 하는데 필요한 능력을 결정하거나 에이전트 집합의 구성을 계획하는데 중요한 것이다.

⊙ 전문적 지식(Expertise)

에이전트에 의해서 표현되는 지식은 얼마나 큰가? 그 크기가 매우 작다면 다중 에이전트를 통합할 필요성이 있음을 보여주는 것이다. 에이전트가 어떤 영역의 전문적 지식을 제공하는가 또는 에이전트가 어플리케이션이나 기업의 어떤 부분을 조정하거나 관리하는데 사용되는 메타지식을 표현하는가? 전문적 지식이 다중 어플리케이션에 사용될 정도로 일반적인 것인가? 에이전트가 실행되는 동안에 상호작용이 가능한가? 만약에 에이전트가 실행되는 동안에 상호작용이 가능하다면 에이전트가 그 내부 프로세스에 문제를 발생시킬지도 모르는 정보를 받으면 어떻게 하는가?

⊙ 상호운영(Interoperability)

이 에이전트가 어플리케이션에 대해서 주문된 대로 쓰여진 것인가? 또는 에이전트가 COTS 소프트웨어나 회사가 개발한 레거시 코드를 포함하고 있는가? 어떤 표현방식, 해석, 프로그램 언어, 운영체제, 그리고 하드웨어 요구조건이 에이전트를 어플리케이션에 내장시키는데 필요한가?

⊙ 동기부여(Motivation)

이 에이전트가 사리(self-interest)를 표출하는 것으로 내부적인 동기를 부여하는가 또는 외부적으로 제어되는가? 만약 내부적으로 동기가 부여된다면 에이전트는 어떻게 사리를 표출하며, 이것을 어떻게 어플리케이션의 제어 범위 내로 통합하는가(예를 들어

서 만약 에이전트가 어떤 경제적인 이득의 모델에 의하여 동기가 부여된다면, 모든 에이전트가 이러한 모델을 공유하는가)? 에이전트가 어떤 작업을 계획하고 실행함으로써 스스로의 움직임을 조정해야 하는가? 다른 에이전트가 그 에이전트를 위하여 작용하기를 기대하는가 혹은 그 에이전트가 다른 인간이나 에이전트를 위하여 서비스를 수행하는가? 만약 그렇다면, 이것들의 관계는 어떻게 정의되는가? 에이전트가 협조적으로 상호작용하는가? 아니면 경쟁적인가? 에이전트가 인간이나 다른 에이전트와 상호작용하는 것이 도발적인가 아니면 반응적인가? 에이전트가 국부적인 목적이나 제한조건을 만족하는 것과 시스템 수준의 목적을 만족시키는 것 사이에서 어떻게 균형을 맞추는가?

⊙ 추상화(Abstraction)

에이전트가 내장되어 있는 환경에 대하여 얼마나 알아야 하는가? 결정을 내릴 때에 다른 에이전트의 능력에 대한 정보를 이용하는가? 상호작용하기 위하여, 에이전트 이름, 물리적 주소, 또는 다른 특정한 정보를 알 필요가 있는가?

⊙ MADS 어플리케이션 특성(MADS Application Characteristics)

다중 에이전트 어플리케이션이 MADS 개발로 인하여 예상되는 장점을 제공하기 위해서는 전체적인 어플리케이션의 설계가 개별 에이전트의 설계와는 구별되어야 한다. 어플리케이션 설계는 에이전트가 상호작용하는 방식과 설계과정에서의 자동화된 상호작용, 협력, 그리고 인간의 참여에 필요한 지원에 그 초점을 맞추게 된다.

특성 설계 어플리케이션의 특성에 의하여 개발과정에서의 선택을 많이 결정해 준다. 이러한 구조는 에이전트가 적당한 시간에 수행될 수 있도록 하는 제어 인프라 구조, 데이터 교환, 공유된 언어와 커뮤니케이션 약정을 촉진시켜주는 기초적인 커뮤니케이션 인프라 구조를 제공해야 한다.

분명히 어플리케이션 인프라 구조는 에이전트의 선택 등과 같은 것에 제한을 주게 된다. 예를 들어서, 스스로의 작업을 계획하고 스스로 정보흐름을 조정하는 인간과 매우 정교한 에이전트가 포함된 어플리케이션은 설계자 사이에서 메시지의 흐름을 주로 지원하는 인프라 구조를 필요로 한다. 이와는 대조적으로, 만약 에이전트가 목표 지향적이거나 협조적인 거동-즉 고립된 프로그램은 직접적으로 불려질 때만 작동한다-에 대해 기본적인 개념이 없는 레가시 도구로 되어 있다면 인프라 구조는 작업을 계획하고 일정을 작성하고, 적당한 시간에 에이전트를 기동하고, 그리고 다양한 에이전트로부터 되돌아온 부분적인 설계를 합치는 등을 포함한 상당한 정도의 기능성을 제공할 필요가 있다. 이러한 기능성은 정교한 구조 내에서는 내장되어지며 또 단순한 구조에서는 컨트롤러로 작용하는 별개의 에이전트에 의하여 제공될 수 있다.

다음에 있는 의문점을 통하여 어플리케이션의 설계를 시작하기 전에 그것의 목표와 요구조건을 도출해 낼 수 있다. 의문점을 안다는 것이 답을 아는 것처럼 잘 정돈된 것은 아니지만 이렇게 함으로써 창의성이나 혁신성을 발휘할 여지를 남기면서도 일종의 설계지원을 제공하는 것이다.

⊙ 상호운영(Interoperability)

에이전트의 이질성을 어느 정도 예상해야 하는가? 에이전트가 다른 프로그램 언어를 사용하고, 데이터를 다른 표현언어와 모델로 표현하고, 다른 컴퓨터 환경에서 작동하고, 제품환경이나 또는 서로의 환경에 대한 가설에 의존하는가 또는 서로간의 메시지를 이해하는가? 상호작용을 하기 위하여 어플리케이션에 해석능력이 필요한가? 만약 그렇다면, 그러한 능력은(그리고 그 어플리케이션은) 자동화되는가?

⊙ 정보흐름(Information Flow)

어떤 종류의 데이터 의존 관계가 에이전트 사이에 존재하며, 그것이 얼마나 밀접하게 같이 작업하며, 그리고 얼마나 많은 정보를 공유할 필요가 있는가? 정보가 한 에이전트에서 다른 에이전트로 자동적으로 움직일 수 있는가 또는 어플리케이션에 인간이 일정 수준 정도로 참여할 필요가 있는가? 어플리케이션이 공유된 데이터나 설계를 공유된 저장소에 저장하는가 혹은 어플리케이션이 필요할 때마다 이 데이터를 지역 데이터베이스에 복제하는가? 정보가 필요한 곳으로 정보를 보내는데 이 구조가 지식 기반 시스템의 도움을 제공하는가 또는 이러한 지식이 에이전트 내에 내장되는가?

⊙ 단기적응성(Short-term Adaptability)

어플리케이션이 연속적이거나 혹은 장기적으로 수행되어야 한다면 에이전트가 동적으로 연결되거나 단절될 필요가 있는가? 예를 들어서, CAD 도구가 전용 컴퓨터에서 수행되고 있다면, 다른 에이전트가 필요할 때마다 연결될 수 있는가?

⊙ 장기적응성(Long Term Adaptability)

어플리케이션이 한번 배치되면 얼마나 오랫동안 안정적이길 기대하는가? 어플리케이션이 에이전트 집합, 제어 방법, 또는 사용된 기술에 있어서의 변화에 쉽게 적응할 수 있어야 하는가? 가장 단순한 설계 시스템도 설계 산업의 역동성과 설계관련기술의 신속한 변화 때문에 상당한 정도의 적응성을 요구할 가능성이 높다.

⊙ 배치(Distribution)

어플리케이션이 어느 정도의 물리적인 배치를 지원하는가: 단일 프로세스, 다중 프로세스, 동일한 플랫폼간의 네트워크, 이종 플랫폼간의 네트워크, 웹 기반?

⊙ 동시성(Concurrency)

에이전트의 작용이 동시적이고 동기적이어야 하는가? 아직도 많은 루틴 설계가 순차적인 프로세스를 이용하고 있지만 요즘의 경향은 하부단계의 필요조건도 미리 고려하는 것이 가능한 동시공학과 같은 방법을 선호하고 있다. 그러나 동시적 작용은 에이전트 사이의 효율적인 실행을 위하여 데이터와 계획의 일관성을 어떻게 유지하는가에 대한 문제점을 몇 가지 발생시킨다. 이것은 다음과 같은 의문점을 제기한다. 모든 에이전트가 가장 최신의 정보를 가지는 것이 얼마나 중요한 것인가? 어플리케이션이 어떻게 정보를 갱신하는가? 중요한 결과를 작동 환경 전반에 걸쳐서 어플리케이션이 어떻게 전파해 주는가? 부분적으로 완성된 설계작업의 경우에 그것과 잠재적으로 관련이 있는 변화가 작동 환경에 발생한다면 어떻게 되는가? 여러 개의 부분적인 설계를 어떻게 합칠 것인가?

⊙ 전략적인 제어(Strategic Control)

어떤 종류의 포괄적인 전략이 설계 프로세스를 제어하는가? 현존하는 설계에 대한 검색, 새로운 설계나 혹은 설계요소의 통합, 설계분석, 제한조건의 전파, 유추추론, 설계공간의 탐구, 대안설계의 평가 등등이 거기에 포함되는가? 포괄적인 전략을 효율적으로 실현하기 위해서 어플리케이션 내에 전략적인 제어를 어떻게 내장하는가? 전략적인 제어에 대한 책임은 어디에 있는가 - 개개의 에이전트에 분산되는가, 인간 사용자에게 분산되는가, 단일한 에이전트나 제어 에이전트의 집합에 집중되는가, 또는 그 구조 내에 직접 내장되는가? 상호 작용이 계획되고, 반사작용적이고, 사용자에 의해 제어되는가 또는 그들의 혼합된 형태인가? 잠재적인 설계의 해법이 어떻게 평가되어지는가? (즉, 한사람이나 하나의 에이전트인가 또는 여러 사람이나 여러 에이전트에 의한 투표인가?)

⊙ 모순조정(Conflict Management)

어플리케이션에 어떤 종류의 모순이 생길 수 있는가? 관여된 에이전트가 항상 참이며 협동적인가? 모순을 해결하는데 어떤 특정한 메카니즘이 사용가능한가? 모순의 조정에 대한 책임은 어디에 있게 되는가? 모순 조정 전략과 상태에 대한 공유된 정보가 공유된 저장소에 있게 되는가 혹은 필요할 때마다 에이전트의 집합 사이에서 지역 데이터베이스로 복제될 것인가?

⊙ MADS를 위한 인프라구조 기술

MADS 어플리케이션의 통합된 인프라 구조를 만드는데 이용되는 기술은 도메인에 독립적이며 협조적인 정보가 지지구조 내에 얼마나 있는가와 또 에이전트 내에 얼마나 있는가에 영향을 미친다. 즉 정보가 저장되고, 공유되는 방식과 에이전트가 교류하는 방식에 영향을 준다. 도메인 수준의 설계에서 발견되는 문제를 반영해주는 경우에는 에이전트를 설계하는 것이 특히 인프라 구조 기술의 선택과 같은 중요한 많은 변수를 매우 일찍 결정하는 것이 필요하다. 이러한 결정을 내리는 시점에 이와 같이 일찍 결정을 내리는 것이 하부단계에 어떤 결과를 미치는지를 아는 것은 매우 어렵다. 다음에 나오는 설명을 통하여 설계자가 MADS 어플리케이션의 인프라 구조를 만드는데 사용한 몇 가지 대안 기술에 대한 간단한 개요를 볼 수 있다. 이것은 포괄적인 목록이 아

니고 다양한 선택에 대하여 자세한 내용을 전달하는 것도 아니다. 그러나 이것은 보다 넓은 관점에서의 구분을 할 수 있게 해준다.

이 인프라 구조의 모든 옵션은 이미 앞의 의문점에서 제기한 기본적인 문제를 다 언급하고 있다. 모든 것은 정보가 공유되도록 해야 한다. 이것은 공유된 지식 표현 방법과 전달 규약 혹은 해석 메카니즘 등을 포함한다. 각각의 기법은 특유의 강점이 있다.

⊙ 에이전트 네트워크(Agent Network)

그림 1은 상호작용하는 두 개의 지역 네트워크를 갖는 에이전트 네트워크를 보여준다. 이와 같은 에이전트 프레임워크에서는 커뮤니케이션이나 상태지식이 통합되어지지 않는다 - 즉 개개의 에이전트가 메시지가 언제 어디로 보내지는가, 다른 어떤 에이전트가 이용가능한가, 다른 에이전트의 능력은 어느 정도인가를 알고 있어야만 한다. 분산된 지능형 설계환경(Distributed intelligent design environment)가 에이전트 네트워크를 이용하는 다중 에이전트 시스템의 예가 될 것이다.[11] DIDE에서는 지역 네트워크에서의 에이전트 사이의 커뮤니케이션은 ToolTalk에 의해 발생하며, 원거리의 커뮤니케이션은 ELM 메일 시스템과 같은 e-mail을 통하여 일어난다. 각각의 에이전트는 그들만의 네트워크 인터페이스가 있는데 이것은 커뮤니케이션 인터페이스, 다른 에이전트의 모델, 지역전문지식, 그리고 현재의 환경에 대한 모델 등이다. 그러나 에이전트 내에 커뮤니케이션이나 제어 전문지식을 내장하는 것은 에이전트 집합에 걸쳐서 정보와 지식을 상당히 중복하는 것이다.

⊙ 연합(Federation)

몇몇의 대형 공학 프로젝트(예를 들어 Palo Alto Collaborative Testbed[1])는 연합이라 불리는 인프라 구조를 사용한다. 연합 시스템에는 활성 데이터를 저장하는 명확한 공유 시설이 없다. 그보다는 이 시스템은 모든 데이터를 지역 데이터베이스에 저장하고 메시지를 교환함으로서 데이터를 갱신하거나 변경한다. 연결성이나 메시지의 경로는 촉진자(facilitator)에 의하여 처리되는데 이것은 다양한 경로규약을 실행

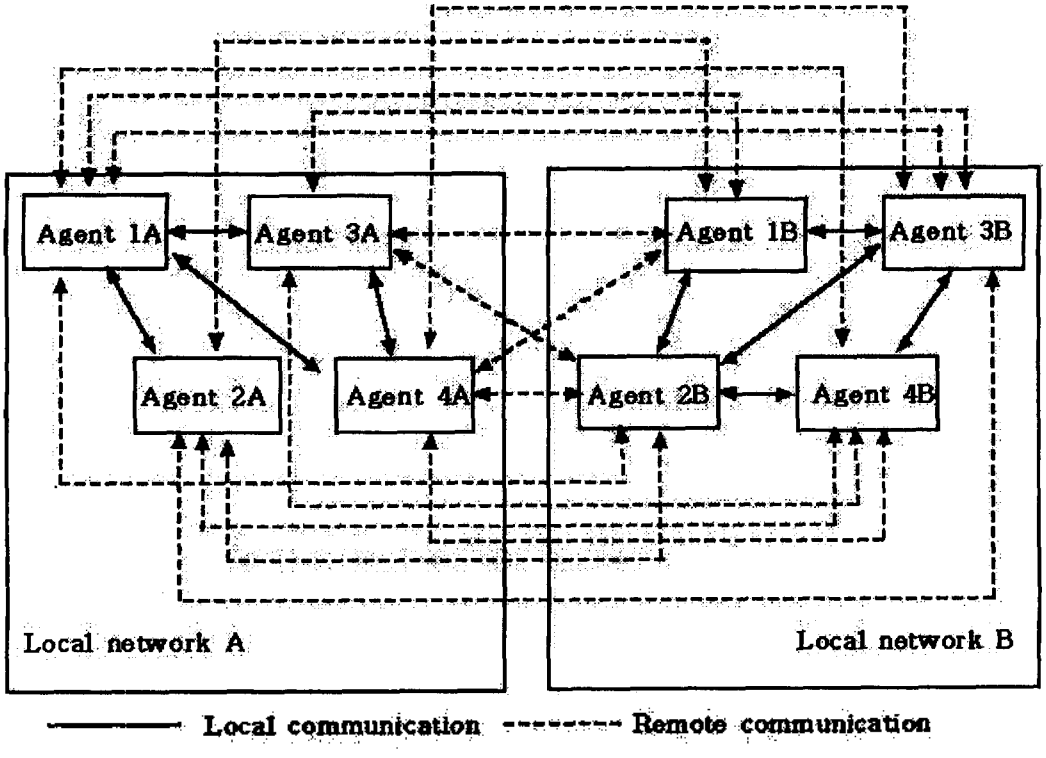

Fig. 1. An agent network.

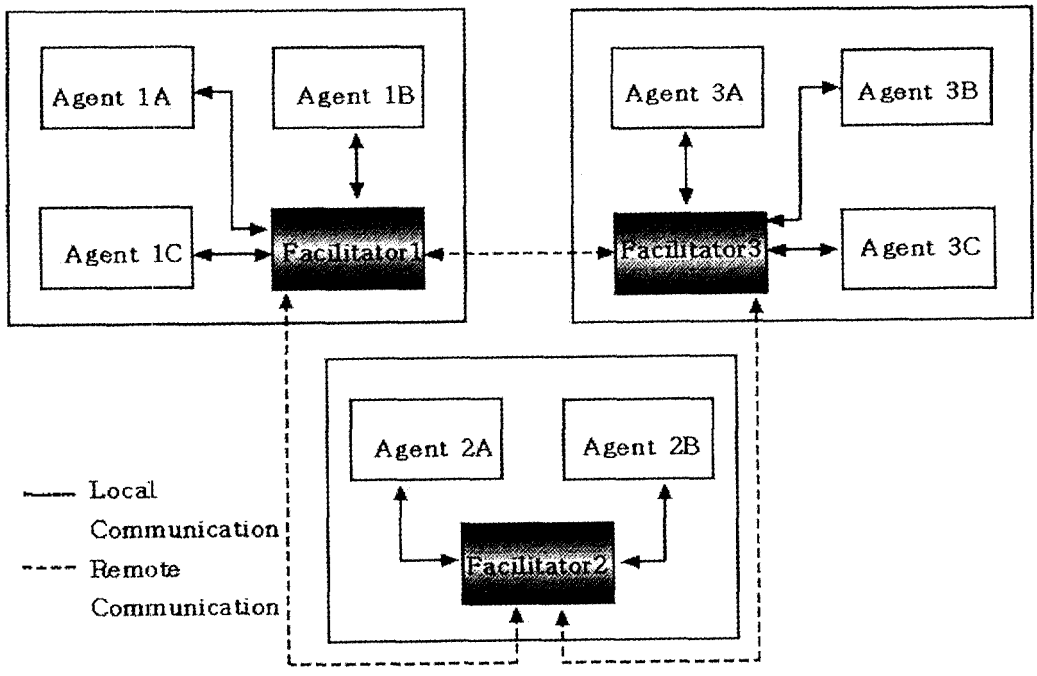

Fig. 2. An agent federation.

하도록 프로그램된 특수 에이전트이다.[12] 촉진자는 어떤 정보나 서비스가 가능한가에 대한 지식의 저장소의 역할을 한다. 에이전트에 서비스가 필요하면 에이전트는 촉진자와 교류하여 촉진자는 이 요구를 다른 에이전트의 능력과 맞추어 준다(지역적일 수도 있고 시스템 전체에 대한 것일 수도 있다).

촉진자 모델은 에이전트가 에이전트 집합에만 특정한 자세한 정보를 알 필요가 없기 때문에 직접 커뮤니케이션 보다 더한 유연성을 갖게 된다. 촉진자를 설치함으로써 화이트 페이지, 옐로우 페이지, 직접 커뮤니케이션, 문제 분리, 이해, 그리고 감시와 같은 고급 수준의 서비스를 갖추게 된다. 그림 2는 에이전트 연합을 보여준다.

⊙ 에이전트 기반 흑판

에이전트 기반 흑판은 연합 시스템과 마찬가지로 에이전트의 상호작용을 수행하는데 그룹핑을 이용한다. 그러나 몇 가지 구조상의 특성 때문에 두 개의 시스템은 구분된다. 그림 3에 보이듯이 에이전트의 각 지역 그룹이 활성화된 공유 데이터를 효과적으로 저장하고 복구하도록 특별히 제공된 데이터 저장소를 공유하고 있다.

에이전트에게 그에 상응하는 작업을 전해준다. 제어셀은 광범위한 협동 전략을 실행하도록 프로그램되어진다. Scott Staley와 Daniel Corkill과 함께 저자는 Ford 자동차 회사의 다중에이전트 설계 프로젝트에 대한 설명으로 에이전트 흑판 어플리케이션을 발표하였다.[2]

MADS의 개발을 수행하는 것은 이론과 소프트웨어에 한계가 있기 때문에 매우 힘드는 일이다. 그러나 에이전트 기반 프로그래밍이 널리 사용되면서 매우 빠르게 변화하는 것이 되었다. MADS를 사용하면서 얻을 수 있는 이득은 새로운 시스템을 개발한다는 것과 보다 중요하게 시스템의 전수명 유지를 하고 에이전트를 구성하는 소프트웨어 요소를 다시 사용하는 것 등으로 말미암아 무시할 수 없는 것이 된다.

MADS가 미래의 설계에서 중요한 역할을 할 것이지만 에이전트 기술을 적용한다고 해서 설계과정에 고유한 복잡성을 해결해 주진 않는다. 사실상 에이

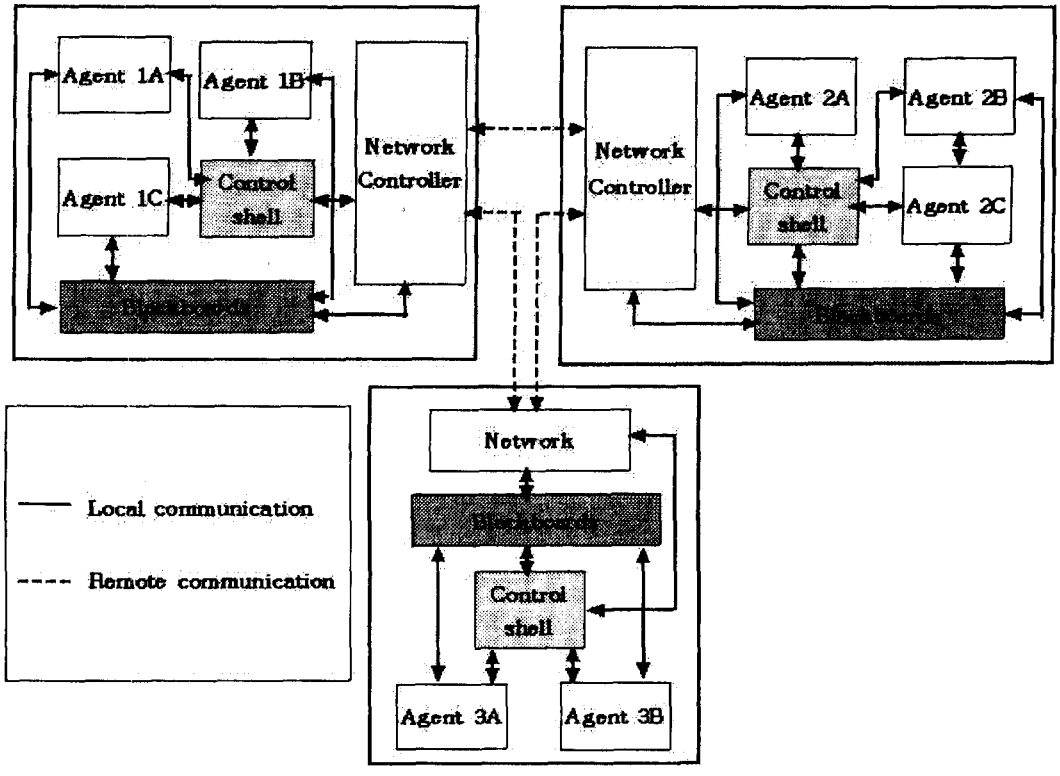

Fig. 3. A distributed blackboard architecture.

전트 기술은 설계과정을 명확하게 표현하지도 않는다. 에이전트의 상호 작업을 통한 정보 흐름의 개선과 자동화는 매우 큰 이득이 될 수 있다. 그러나 정보를 잘못된 시간에 잘못된 곳으로 보냄으로서 시스템이 혼동되거나 산만하게 된다. 이와 마찬가지로 에이전트의 설계작용을 자동화하고 에이전트 내에 목적 지향의 제어능력을 내장하는 것이 설계과정을 빠르게 할 수는 있지만 그 결과는 잘못된 일을 잘못된 시간에 더 빨리 한 것이 될 수도 있는 것이다. 효과적인 공동작용과 모순조정모델과 협동의 기술을 개발하는 것이 혁신적인 기술에 필요한 결정적인 링크가 된다.

감사의 글

이 연구는 미 국방성의 Advanced Research Projects Agency(ARPA) Manufacturing Automation and Design Engineering(MADE) RaDeo No. 70NANB6H0074 협약에 의해 부분적인 지원을 받았다.

References

1. M. R. Cutkosky *et al.*, "PACT: An Experiment in Integrating Concurrent Engineering Systems," Computer, **26**(1): 28-38 (1993).
2. S. E. Lander, S. M. Staley, and D. D. Corkill. "Designing Integrated Engineering Environments: Blackboard-Based Integration of Design and Analysis Tools," Concurrent Engineering: Research and Applications, special issue on the application of multiagent systems to concurrent engineering, **4**(1): 59-72 (1996).
3. C. M. Pancerella, A. J. Hazelton, and H. R. Frost, "An Autonomous Agent for On-Machine Acceptance of Machined Components," Proc. Modeling, Simulation, and Control Technologies for Manufacturing, Int'l Soc. for Optical Eng., Bellingham, Wash., pp. 146-159 (1995).

4. J. Lin, M. S. Fox, and T. Bilgic, "A Requirement Ontology for Engineering Design," Concurrent Engineering: Research and Applications, 4(3): 279-291 (1996).
5. T. Finin *et al.*, "KQML as an Agent Communication Language," Proc. Third Int'l Conf. Information and Knowledge Management, ACM Press, New York, 1994.
6. T. J. Mowbray and R. Zahavi, The Essential CORBA: Systems Integration Using Distributed Objects, johnWiley & Sons, New York, 1995.
7. S. E. Lander and V. R. Lesser, "Sharing Meta-Information to Guide Cooperative Search among Heterogeneous Reusable Agents," IEEE Trans. Knowledge and Data Engineering, to appear in 1997.
8. M. Klein, "Supporting Conflict Resolution in Cooperative Design Systems," IEEE Trans. Systems, Man, and Cybernetics, 21(6): 1379-1390 (1991).
9. K. J. Werkman, "Multiple Agent Cooperative Design Evaluation Using Negotiation," Artificial Intelligence in Design, J. S. Gero and F. Sudweeks. eds., Kluwer Academic Publishers, London, 1992.
10. T. P. Darr and W. P. Birmingham, "Automated Design for Concurrent Engineering," IEEE Expert, 9(5): 35-42 (1994).
11. W. Shen and J. P. Barthes, "DIDE: A Multiagent Environment for Engineering Design," Proc. First Int'l Conf. Multiagent Sysems, AAAI Press, Menlo Park, Calif., pp. 344-351 (1995).
12. M. R. Genereth and S. P. Ketchpel, "Software Agents," Comm. ACM. special issue on intelligent agents, 37(7): 48-53 (1994).

원 저자인 Susan E. Lander는 Amherst의 Blackboard Technology의 책임연구원이다. 그녀의 관심연구분야는 협조적인 에이전트 기반시스템, 에이전트 기반 공학설계, 분산검색, 지식공유, 그리고 에이전트시스템의 모순관리에 흑판기법을 사용하는 것이다. 그녀는 University of Massachusetts, Amherst의 전산학과에서 학사, 석사, 박사를 취득했다. 그녀는 AAAI, ACM, IEEE의 회원이다. 관심있는 독자들은 다음의 주소를 통하여 Lander와 연결될 수 있다. Blackboard Technology, Amherst, MA 01002; lander @bbtech.com; http://www.bbtech.com/lander/

≪IEEE Expert Intelligent System & Their Application Vol. 12, No. 2, April 1997≫

본 기사는 경희대의 김영진 편집위원이 "IEEE Expert Intelligent System & Their Application"에서 발췌하였으며 출판사인 IEEE Computer Society의 연락처는 다음과 같다.

• http://www.computer.org/pubs/expert/expert.htm