지능형 로봇 제어를 위한 시스템 통합 방법론

서울시립대학교 곽별샘 · 이재호

1. 서 론

특정한 목적만을 수행하기 위해 하나의 작업에 전문화된 로봇을 개발하던 로봇 산업은 로봇이 처한 환경의 변화에 민감하게 반응하고 적절히 대처하며 다양한 목적을 수행할 수 있는 지능형 로봇을 구현하기위한 방향으로 급속히 발전하고 있다. 이제는 로봇을 위해 환경을 제약하던 과거와 달리 로봇 스스로 환경의 변화를 감지하고 변화에 적절히 대응해야 하며 따라서 로봇이 같은 목적을 수행하더라도 감지된 상황의 변화에 따라 다른 행동을 보일 수 있어야한다[6]. 이를 위해 로봇 시스템은 환경의 변화를 민감하게 인식하고 다양한 행위를 취하기 위해 많은 기술들을 이용한다.

지능형 로봇 시스템은 다양한 로봇 기술들을 효과적으로 통합해야 한다. 많은 기술들이 적절한 통합 방법을 거치지 않은 채 통합되어 운용되면 각 기술들이 강하게 결합되어 유지보수가 힘들 뿐 아니라 새로운 기술의 추가를 힘들게 한다. 로봇의 각 기능을 효과적으로 통합하고 운용하기위한 다양한 통합 방법론이 존재한다. 예를 들어 계층 구조를 이용하면 시스템 구성 요소간의 결합력을 낮추고 계층 내부의 구현을 숨기기 때문에 유지보수 및 관리가 쉽다. 또한 상위 계층으로의 추상화를 통해 복잡성을 낮추어 상위 계층의 작업량을 줄이는 효과가 있다. 통합 방법의 또 다른 예로 블랙 보드 구조[8]가 있다. 블랙 보드 구조는 중심에 정보 저장소를 두고 이것을 통해 각각의 구성 요소가 정보를 주고받으면서 실행하도록 구성된다. 이것은 각기 다른 전문화된 구성 요소간의 통합을 용이하게 한다. 각 구성 요소는 제어를 위해 필요한 정보를 주고받을 뿐 직접 제어하지 않기 때문에 구성 요소 사이의 결합력은 거의 없다고 볼 수 있다. 따라서 각 구성 요소의 유지보수 및 새로운 기능의 추가가 쉽게 되는 장점을 갖는다.

로봇 시스템의 통합은 그 대상을 로봇 내부의 구성 요소에서 웹으로 확장할 수 있다. 웹 서비스의 통합을 통해 로봇은 유비쿼터스 환경[9]을 이해할 수 있고 홈 네트워크 등과의 연동을 통해 보다 지능적으로 다양한 목적을 수행할 수 있을 것이다.

본 논문에서는 로봇의 각 기능 요소들을 쉽게 통합하고 제어할 수 있는 로봇 시스템 통합 방법을 제시한다. 제시하는 로봇 시스템 통합 방법에서는 로봇 시스템을 3-계층으로 나누어 행위에 대한 제어를 추상화하며, 최상위 계층에 지능형 에이전트를 이용하여 환경의 변화를 인식하고 적응하여 주어진 목적을 달성하기 위한 다양한 행위의 조합을 찾아내는 작업 관리기를 정의한다. 효과적인 시스템 통합을 위해 제어 컴포넌트를 정의하고 제어 명령과 제어 방법을 정의하며 분산 환경을 고려한다.

2. 효과적인 시스템 통합 방법의 필요성

로봇의 각 기능을 효과적으로 통합하고 관리하기 위해서는 다음과 같은 사항이 필요하다.

- **개방성: 표준적이며 개방적 인터페이스를 통하여 다양한 기술과 접목될 수 있어야 한다.** 지능형 로봇의 실현을 위해서는 방대한 기술을 구현한 소프트웨어 컴포넌트들이 효과적으로 결합되어야 한다. 이러한 다양한 컴포넌트들이 표준적이고 개방적 인터페이스를 가져야만 다양한 기술 개발자들이 참여하여 효과적인 시스템 통합을 이룰 수 있다. 로봇의 각 요소가 서로 강하게 결합되어 그것의 기능이 로봇 내부로만 한정되어버리면 각 요소가 다른 분야로 응용될 수 있는 가능성을 좁힐 수 있다. 또한 이것은 역으로 다른 분야에서 사용 중인 기술 및 기능을 로봇에 적용시킬 때에도 문제점으로 작용한다.
- **확장성: 새로운 기능을 추가하거나 개선하는 것이 용이하고 기존의 시스템에 미치는 영향이 최소화되어야 한다.** 지능형 로봇은 특성상 새로운 기능의 추가나 개선이 지속적으로 필요하며 이에 따라 로

봇 기능이 확장성이 중요하다. 로봇의 기능 요소들과 이것을 제어하는 로직이 강하게 결합되어 있으면 한 요소의 수정에 대한 파급 효과가 매우 크다. 이로 인해 시스템을 유지하고 보수하는 데는 것이 어렵고 많은 비용이 소모된다. 또한 이러한 영향은 시간이 지남에 따라 시스템의 안정성을 떨어뜨리는 결과를 초래한다.

- **호환성: 로봇의 기능 요소 단위로 교체되거나 다른 로봇과 공유될 수 있어야 한다.** 로봇의 각 요소가 서로 강하게 결합되면 로봇의 기능 요소 단위로의 교체가 어렵게 된다. 나아가 하나의 지능형 로봇에서 적용되거나 습득된 기술을 다른 로봇과 공유하기 위해서는 기능 요소 간의 호환성이 요구된다. 이를 위해서는 역시 개방적 표준적 인터페이스가 필요하게 된다.

로봇을 구성하는 각 요소가 밀접하게 결합되어 있으면 위의 요구사항을 만족시키기 어렵게 된다. 밀접한 결합은 새로운 구성 요소의 추가나 이미 존재하는 구성 요소의 유지보수, 제거 및 관리 차원에서 많은 어려움을 발생시킨다. 따라서 로봇을 구성하는 각 요소간의 결합을 적절히 분리할 필요가 있으며 기능의 추가, 제거, 유지보수 등에 대해 유연하게 대처할 수 있어야 한다.

로봇의 각 구성 요소를 효과적으로 통합하고 유지, 보수 및 관리의 효율성을 높이기 위해서 3장에서는 3-계층 아키텍처를 설명하고, 4장에서는 통합 관리기를 제시한다.

3. 3-계층 아키텍처

많은 로봇 시스템이 그림 1과 같은 3-계층(Layer) 아키텍처를 채택하고 있다. 계층 아키텍처는 위로 올라갈수록 제어 방법을 추상화하기 때문에 제어 계층은 제어를 위한 복잡한 설정보다는 제어 그 자체에 집중할 수 있다. 이는 뒤에 설명할 작업 관리기가 적절한 행위를 추론하고 조합하는 것을 수월하게 한다.

3-계층 아키텍처는 로봇의 물리적인 장치들을 다루기 위한 디바이스 드라이버와 반응 작용 처리 루틴 등을 수행하는 리액티브 계층(reactive layer), 이러한 리액티브 계층의 인터페이스를 이용하여 실제로 행위를 취하는 시퀀싱 계층(sequencing layer), 그리고 마지막으로 환경에 적응하며 주어진 목적을 달성하기 위해 시퀀싱 계층의 컴포넌트들의 행위를 조합하고 조작하는 딜리버레이트 계층(deliberate layer)으로 이루어진다.

딜리버레이트 계층에서는 로봇의 최상위 의사 결정

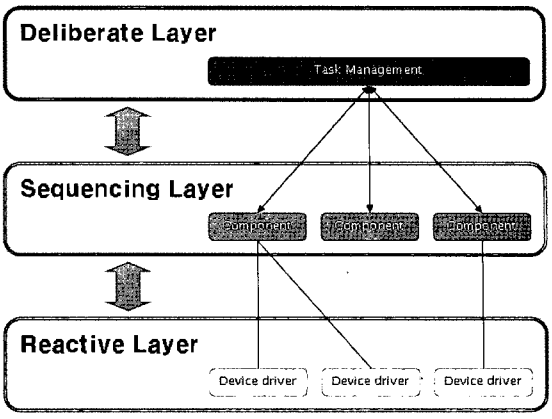

그림 1 일반적 3-계층 아키텍처

이 이루어진다. 이 계층은 내부로부터 생산된 정보와 시퀀싱 계층으로부터 전달된 정보를 가공하여 행위의 판단 근거로 삼으며, 적절한 판단을 위한 판단 법칙을 포함한다. 결정된 최상위 의사 결정은 현재 상황에서 가장 적절히 수행될 수 있도록 행위의 조합을 생성하며 시퀀싱 계층이 제공하는 제어 인터페이스를 통해 행위를 취한다. 행위를 취하는 도중에도 로봇 내외부의 정보를 계속 수집·가공하며 진행 중인 행위에 대한 감시를 계속한다. 환경의 변화로 인해 현재 취하고 있는 행위가 부적절하다고 판단되는 경우 행위를 취소하고 새로운 행위의 조합을 찾아내거나 현재 행위가 변화된 환경에 적응할 수 있도록 적절한 조취를 취한다.

시퀀싱 계층은 로봇의 각 기능을 서비스하는 컴포넌트들로 이루어진다. 이 계층의 컴포넌트들은 리액티브 계층을 이용하거나 독자적인 소프트웨어 컴포넌트로 구현된다. 시퀀싱 계층은 이러한 컴포넌트들의 등록, 검색, 관리 등의 기능을 지원한다.

리액티브 계층은 로봇의 하드웨어와 밀접하게 관계가 있다. 하드웨어를 제어하는 디바이스 드라이버들이 위치하며 시퀀싱 계층에게 제어 인터페이스를 제공한다. 일반적으로 로봇의 구동은 딜리버레이트 계층의 판단과 행위의 결정이 시퀀싱 계층을 거쳐서 리액티브 계층으로 전달되며, 리액티브 계층에서 감지된 정보는 시퀀싱 계층을 거쳐 딜리버레이트 계층으로 전달된다. 이렇게 전달된 정보는 딜리버레이트 계층이 작업 수행에 대한 판단 자료로 활용된다. 그러나 갑작스런 환경의 변화는 이것을 딜리버레이트 계층까지 전달하고 판단을 기다리기까지 충분히 기다릴 수 없으므로 이때에는 리액티브 계층에서 스스로 판단하고 즉시 대처하며 대처 결과를 딜리버레이트 계층에게 전달한다.

4. 통합 관리기의 구조

통합 관리기는 로봇의 3-계층 중에서 딜리버레이트

계층에 속한다. 로봇의 최상위 의사 결정을 담당하며 결정된 의사를 수행하기 위해 시퀀싱 계층의 컴포넌트들을 적절히 조합하고 제어한다. 통합 관리기의 전체적인 구조는 그림 2와 같다.

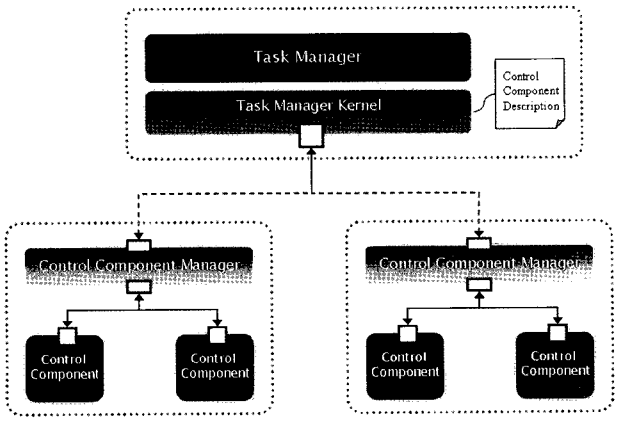

그림 2 통합 관리기의 구조

• 작업 관리기(Task Manager)

구조 최상단에 위치한 작업 관리기는 로봇 내외부의 환경의 변화에 대한 인식 결과를 종합하고 그것을 판단의 근거로 활용한다. 로봇의 목적을 수행하기 위해 인식 결과를 충분히 활용하고 그것에 기반을 두어 목적을 달성할 수 있는 제어 컴포넌트들의 조합을 찾아 실행 상태를 제어한다. 작업 관리기는 환경의 변화를 인지하고 그것에 의해 현재 취하고 있는 행위가 부적절하다고 판단되면 행위를 중단하고 현재 상황에 맞는 행위의 조합을 재검색하거나 변화된 환경에 적응할 수 있도록 행위에 적절한 조취를 취한다.

• 작업 관리 커널(Task Manager Kernel)

작업 관리 커널은 작업 관리기에게 제어 컴포넌트를 제어할 수 있는 권한을 부여하고 제어 인터페이스를 제공한다. 모든 제어 컴포넌트의 실행 상태를 종합하여 모니터링하며 작업 관리기와 제어 컴포넌트 사이의 제어 요청과 응답, 정보 전달을 연계한다.

• 제어 컴포넌트 매니저(Control Component Manager)

제어 컴포넌트 매니저는 그것이 실행중인 기계 내의 제어 컴포넌트를 관리한다. 작업 관리기의 요청에 의해 제어 컴포넌트를 메모리에 적재하며 실행 중인 제어 컴포넌트의 실행 상태를 모니터링한다.

• 제어 컴포넌트(Control Component)

제어 컴포넌트는 작업 관리기로부터 제어를 받는 컴포넌트로써 작업 관리기가 바라보는 기능의 최소 단위이다. 하나의 제어 컴포넌트는 로봇의 기능 요소 하나 이상을 연계하여 작업 관리기의 최소 기능 단위를 구성한다. 작업 관리기로부터 제어 요청을 받아 실행 상태를 변경하며 작업 관리기에 필요한 정보를 제공한다.

작업 관리기가 제어 컴포넌트만 제어하기 때문에 로봇의 각 기능 요소들에 대한 결합력을 낮출 수 있다. 게다가 작업 관리기와 제어 컴포넌트가 네트워크로 분리되어있기 때문에 로봇 시스템을 단일 하드웨어로 제한하는 것을 방지하고 로봇의 각 기능을 로봇 외부로 분산시킬 수 있게 한다. 이러한 구조는 다음과 같은 장점을 갖는다.

• 로봇의 새로운 기능 요소를 쉽게 추가할 수 있다.

새로운 하드웨어나 소프트웨어 컴포넌트의 추가 등으로 새로운 기능 요소를 추가해야할 경우 그것을 활용할 제어 컴포넌트를 수정하거나 제어 컴포넌트화함으로써 로봇 시스템에 쉽게 추가할 수 있다.

• 기능 요소의 유지 보수가 쉽다.

로봇의 각 기능 요소는 딜리버레이트 계층에 직접적인 영향을 미치지 않는다. 기능 요소의 유지 보수에 대한 파급 효과는 그것을 둘러싼 제어 컴포넌트로 제한된다.

• 분산 환경을 지원함으로써 로봇의 자원을 효율적으로 활용할 수 있다.

로봇이 이용 가능한 하드웨어를 적극 활용하여 제어 컴포넌트를 적절히 분배함으로써 로봇의 자원을 효율적으로 활용하고 이를 통해 로봇의 각 기능 요소를 최대한으로 활용할 수 있게 한다.

• 로봇의 기능을 로봇 외부로 확장할 수 있다.

제안하는 통합 관리 구조는 분산 환경을 지원하므로 로봇 외부의 기능을 로봇 기능의 일부로 사용할 수 있다.

다음 장에서는 통합 관리기의 각 구성 요소를 구체적으로 정의하고 그것들이 어떻게 상호 작용하는지 설명한다.

5. 통합 관리기의 각 구성 요소

5.1 작업 관리기

작업 관리기는 로봇에게 주어진 목적을 달성하기 위해 제어 컴포넌트를 적절히 조합하고 작업 관리 커널이 제공하는 인터페이스를 통해 제어 컴포넌트를 조작한다. 제어 컴포넌트의 실행 상태, 제어 컴포넌트로부터 제공된 정보 등을 통해 로봇 내외부의 상황을 인지하여 그것을 판단의 근거로 활용하며, 상황의 변화에 따라 제어 컴포넌트의 실행 상태를 조작하여 변화에 능동적으로 대처한다. 이렇게 주어진 상황을 인지하고 능동적으로 판단하여 상황의 변화에 적절히 대응하기 위해 작업 관리기는 지능형 에이전트로 구현하며 본

논문에서는 JAM 에이전트 시스템을 이용해 작업 관리기를 구현했다.

BDI 아키텍처를 기본으로 하고 있는 JAM 에이전트 시스템은 그림 3에서과 같이 실세계 모델(World Model), 실행 계획 라이브러리(Plan Library), 주어진 목적(Goal), 의도 구조(Intention Structure)로 구성되어 있다[3, 5].

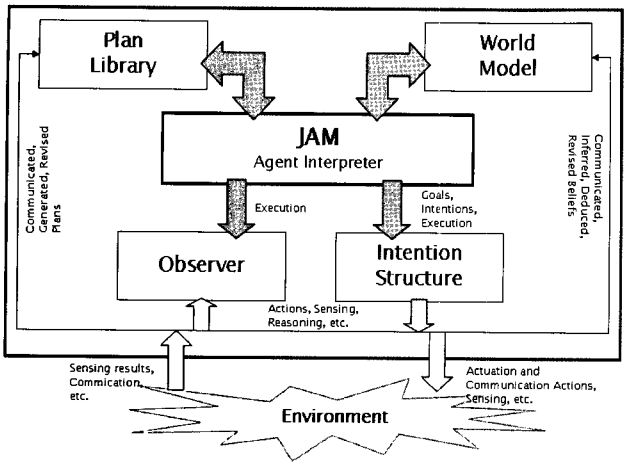

그림 3 JAM 에이전트 시스템의 구조

로봇에게 주어진 일 또는 목적은 JAM 에이전트 시스템의 목적(Goal)으로 표현되고, 로봇의 내외부의 상황은 실세계 모델에 반영된다. JAM 에이전트 시스템은 실세계 모델을 판단의 근거로 주어진 목적을 달성할 수 있는 적절한 실행 계획을 검색한다. 실행 계획에는 특정 목적을 달성할 수 있는 일련의 작업 절차와 그것이 수행되기 위해 필요한 선행 조건(Precondition), 수행을 유지하기 위한 조건(Context), 실패 시 행동 절차 등이 서술되어 있으며, 이 경우 작업 절차로써 적절한 제어 컴포넌트의 조작 방법을 서술한다. 실행 계획은 일련의 작업 절차를 서술하는 것뿐만 아니라 새로운 목적을 생성하거나 세부 목적을 생성할 수 있어서 실행 시간에 보다 많은 실행 계획의 조합이 이루어질 수 있다. 이러한 지능형 에이전트 시스템을 통해 보다 지능적인 로봇의 제어가 가능하다.

5.2 제어 컴포넌트 배포 설명서(Description)

제어 컴포넌트는 시스템이 시작된 후 선택적으로 메모리에 적재되며 따라서 시스템은 제어 컴포넌트가 메모리상에 하나도 적재되지 않은 상태에서 시작된다. 따라서 분산 환경 내에서 작업 관리기 및 작업 관리 커널이 특정 제어 컴포넌트를 지목하기 위해서는 제어 컴포넌트의 배포 상황을 설명할 수 있는 제어 컴포넌트 배포 설명서(Control Component Description)가 필요하다.

제어 컴포넌트 설명서에는 제어 컴포넌트 매니저를 지칭하는 이름과 물리적인 위치, 그리고 그것이 관리할 수 있는 제어 컴포넌트들의 배포 관련 설명이 포함된다. 다음은 제어 컴포넌트 설명서의 예이다.

```
<ControlComponentDescription>
  <ControlComponentManagerList>
    <ControlComponentManager name="LocalJavaCCManager"
          ipAddress="localhost" port="4004">
      <ControlComponent name="VoiceExpression"
          classpath="example.VoiceExpression"/>
      <ControlComponent name="FaceExpressionRecognizer"
          classpath="example.FaceExpressionRecognizer"/>
    </ControlComponentManager>
  </ControlComponentManagerList>
</ControlComponentDescription>
```

5.3 작업 관리 커널

작업 관리 커널은 제어 컴포넌트 배포 설명서를 통해 제어 컴포넌트의 물리적 배포 상황을 파악하고 작업 관리기에게는 그것을 지칭하는 이름으로 접근할 수 있는 제어 인터페이스를 제공한다. 작업 관리기는 제어 컴포넌트의 물리적인 배포 현황에 관계없이 제어 컴포넌트의 이름만으로 접근하므로 제어 컴포넌트의 배포 상황이 변경되어도 작업 관리기의 제어 로직에 전혀 영향을 미치지 않는다. 작업 관리기에게 제어 컴포넌트의 물리적인 배포 상황을 숨김으로써 작업 관리기와 제어 컴포넌트 사이의 결합을 한 단계 느슨하게 만든다. 따라서 제어 컴포넌트의 배포 상황을 쉽게 변경할 수 있으며 새로운 제어 컴포넌트의 등록은 물론 특정 제어 컴포넌트를 다른 제어 컴포넌트로 교체하는 것이 쉽게 가능하다. 작업 관리 커널은 또한 제어 컴포넌트 관리자로부터 받은 제어 컴포넌트의 실행 상태를 종합하여 모든 제어 컴포넌트의 실행 상태를 통합 관리한다.

5.4 제어 컴포넌트 관리자

제어 컴포넌트 관리자는 제어 컴포넌트를 메모리에 적재하고 적재된 제어 컴포넌트의 실행 상태를 모니터링하며, 작업 관리 커널과 함께 제어 컴포넌트와 작업 관리기 사이의 통신을 담당한다.

제어 컴포넌트 관리자는 제어 컴포넌트를 메모리에 적재하는 기능을 수행하지만 메모리에 적재된 제어 컴포넌트의 실행 상태를 직접 변경하지는 않는다. 작업 관리기로부터의 제어 요청은 그대로 제어 컴포넌트에게 전달되며 제어 컴포넌트가 그것을 수락함으로써 실행 상태가 변경된다.

실행 중인 제어 컴포넌트는 제어 컴포넌트 관리자에 의해 모니터링된다. 제어 컴포넌트의 실행 상태가 변경되면 제어 컴포넌트 매니저는 작업 관리 커널에게 이를 보고한다. 제어 컴포넌트가 오류 등으로 인해 비정

상적으로 종료하게 되면 오류 상황을 작업 관리기에게 보고하지만 이렇게 종료된 제어 컴포넌트를 재실행하는가에 대한 결정은 작업 관리기가 판단하므로 제어 컴포넌트 관리자의 역할은 오류 상황을 보고하는 것으로 제한한다.

5.5 제어 컴포넌트

제어 컴포넌트는 로봇의 작업 관리기가 바라보는 최소 단위 기능이다. 작업 관리기는 로봇 내부의 세세한 기능 요소에 관여하지 않으며 오직 제어 컴포넌트만을 이용해 로봇을 제어하게 된다. 제어 컴포넌트는 그림 4와 같은 실행 상태를 갖는다.

제어 컴포넌트는 최초 메모리에 적재되지 않는다. 시스템의 자원을 효율적으로 관리하기 위해 작업 관리기에 의해 필요시 메모리에 적재/해제된다. 따라서 UNLOADED 상태는 제어 컴포넌트 내부의 상태가 아니라 제어 컴포넌트 관리자에 의해 관리되는 상태이다.

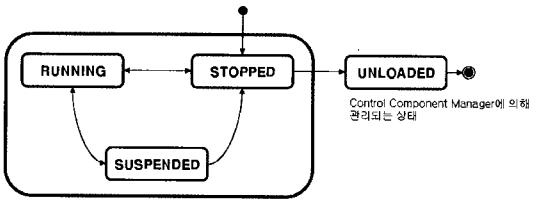

그림 4 제어 컴포넌트의 실행 상태

작업 관리기의 실행 요청에 의해 제어 컴포넌트는 실행되며 실행 중인 상태를 RUNNING으로 정의한다. 제어 컴포넌트의 실행 상태 변화는 즉각 작업 관리기 커널에 반영되며 작업 관리기가 그것을 인식한다. 제어 컴포넌트의 실행이 완료되면 실행 상태는 STOPPED가 된다. 작업 관리기의 판단에 의해 실행 중인 제어 컴포넌트를 정지시키면 제어 컴포넌트는 실행을 멈추고 STOPPED가 된다. 또는 작업 관리기의 판단에 의해 일시 정지시킬 수도 있으며 이 경우엔 SUSPENDED가 된다. 일시 정지 중에도 작업 관리기에 의해 실행을 정지시킬 수 있다.

작업 관리기에 의한 제어 컴포넌트의 제어는 요청과 응답으로 이루어진다. 이것은 다시 말해 작업 관리기가 제어 컴포넌트를 바로 정지시키는 것이 아니라 정지시킬 것을 요청하고 제어 컴포넌트가 이것을 수락함으로써 정지됨을 말한다. 이것은 즉각적인 수행 중단이 기능 요소의 내부 상태를 불안정하게 만드는 것을 방지한다. 따라서 제어 컴포넌트는 작업 관리기로부터 받은 요청을 수락할 수 있는 시점을 명시적으로 표현할 수 있는 방법을 제공하며 개발자는 쉽게 이것을 이용할 수 있다. 제어 컴포넌트를 구현함은 다음의 메소드를 구현함을 뜻한다.

```
abstract void execute(parameter: String)
```

작업 관리기의 실행 요청에 의해 제어 컴포넌트의 execute() 메소드가 실행된다. execute()가 수행중인 상태를 RUNNING, 수행이 완료되거나 정지되면 STOPPED로 정의한다.

실행 중인 제어 컴포넌트가 작업 관리기로부터의 제어 요청을 수락하기 위해 개발자는 적절한 시점에 다음의 메소드를 execute() 내에서 명시적으로 호출한다.

```
void tmCheckPoint()
```

execute() 내부에서 수행하는 도중 작업 관리기의 제어 요청을 수락해도 괜찮은 지점에 명시적으로 tmCheckPoint()를 호출한다. 이 때 작업 관리기로부터 일시 정지 요청이 있었다면 tmCheckPoint() 메소드는 실행 재개 요청이 올 때까지 리턴되지 않는다. 또는 작업 관리기로부터 정지 요청이 있었다면 tmCheckPoint()를 호출함과 동시에 수행이 정지될 것이다.

execute() 메소드 내에서 로봇의 각 기능 요소를 적절히 사용함으로써 기능 요소를 제어 컴포넌트화할 수 있다. 또한 작업 관리기에 의한 실행 제어가 즉각적으로 반응하지 않고 요청을 수락할 수 있는 시점을 명시할 수 있기 때문에 시스템을 보다 안정하게 유지할 수 있다. 제어 컴포넌트의 실행 상태가 변경되는 시점에서 보다 안정성을 확보하기 위해 부수적으로 다음과 같은 메소드를 구현할 수 있다.

```
void onStoppedToRunning(parameter: String)
void onRunningToStopped(parameter: String)
void onRunningToSuspended(parameter: String)
void onSuspendedToRunning(parameter: String)
void onSuspendedToStopped(parameter: String)
```

이름에서 알 수 있듯이 XXX 실행 상태에서 YYY 실행 상태로 변경될 때 onXXXToYYY() 메소드가 호출된다. 예를 들어 onStoppedToRunning()은 작업 관리기의 실행 요청에 의해 제어 컴포넌트가 실행되기 직전에 호출되고, tmCheckPoint()가 실행되는 시점에서 제어 컴포넌트로 일시 정지 요청이 있었으면 onRunningToSuspended()가 호출되고 실행 재개 요청이 올 때까지 대기 상태로 들어간다. onXXXToYYY()를 구현하는 것은 개발자의 선택이며 필요하지 않은 경우 구현할 필요는 없다.

앞서 제어 컴포넌트는 작업 관리기에 의해 메모리에 적재/해제된다고 했다. 적재/해제되는 시점에서도 부수적인 작업을 처리할 수 있도록 위와 비슷한 메소드를 제공한다.

```
void onLoad()
void onUnload()
```

작업 관리기에 의해 메모리에 적재된 제어 컴포넌트는 적재 직후 onLoad()가 호출되고, 메모리에서 해제되는 제어 컴포넌트는 onUnload()가 호출되고 나서 메모리에서 해제된다. onXXXToYYY() 메소드와 마찬가지로 구현은 개발자의 선택이며 필요하지 않은 경우 구현할 필요는 없다.

제어 컴포넌트는 경우에 따라 작업 관리기에게 특정 정보를 제공할 수 있다. 작업 관리기에게 정보를 제공하기 위해 다음의 메소드를 제공한다.

```
void tmNotify(notification: String)
```

예를 들어 주행을 담당하는 제어 컴포넌트가 수행 중에 장애물에 직면했을 경우 작업관리기에 알리기 위해 tmNotify()를 호출한다.

제어 컴포넌트는 작업 관리기의 실행 요청에 의해 실행되며 실행에 필요한 파라미터를 작업 관리기로부터 제공 받지만 수행 중에도 작업 관리기로부터 추가적인 정보를 받아야 하는 경우가 있다. 작업 관리기로부터 실행 중 필요한 정보를 받아 그것을 처리하기 위해 다음의 메소드를 제공한다.

```
void onInformed(parameter: String)
```

작업 관리기가 실행 중인 제어 컴포넌트에게 추가 정보를 제공하면 제어 컴포넌트의 onInformed() 메소드가 호출되며 그것의 인자로 해당 정보가 전달된다.

6. 메시지 기반의 제어 요청 및 정보 전달

앞서 살펴보았듯이 작업 관리기에 의한 제어 컴포넌트의 실행 제어는 제어 컴포넌트로의 제어 요청과 그것으로부터의 수락으로 이루어진다. 이러한 비동기적인 제어 방식을 통해 작업 관리기가 제어 컴포넌트의 실행 상태가 변할 때까지 대기 상태에 빠지는 것을 피할 수 있다. 따라서 작업 관리기는 필요한 경우 제어 컴포넌트에게 제어 요청을 전달하고 다른 작업을 수행할 수 있게 된다. 또한 메시지를 통한 제어 요청과 정보 전달은 메시지로 분리된 시스템이 서로 다른 플랫폼에서 운용되는 것을 가능하게 한다. 좁게는 각각이 다른 언어로 개발될 수 있게 하고 넓게는 각각이 다른 시스템에서 운용될 수 있게 한다.

앞서 제어 컴포넌트의 실행 상태를 정의했으므로 여기서는 실행 상태를 제어할 수 있는 제어 명령을 정의한다.

LOAD	제어 컴포넌트를 메모리에 적재
START	실행을 요청
SUSPEND	일시 정지를 요청
RESUME	실행을 재개
STOP	정지 요청
UNLOAD	메모리에서 제거

UNLOAD 명령을 제외한 나머지 명령들은 제어 요청을 구체적으로 설명할 수 있는 정보를 포함한다. 예를 들어 START 명령의 경우 제어 컴포넌트의 실행에 필요한 부가 정보 등을 포함한다. LOAD와 UNLOAD 명령은 작업 관리기로부터 제공되는 정보가 없다. 그러나 LOAD 명령의 경우, 작업 관리 커널에 의해 제어 컴포넌트의 물리적인 위치 정보가 포함된다.

작업 관리기는 작업 관리 커널이 제공하는 제어 인터페이스를 호출함으로써 제어 컴포넌트에게 제어를 요청한다. 이 때 제어 요청은 작업 관리 커널에 의해 제어 메시지로 변환되며 적절한 제어 컴포넌트 매니저에게 전달된다. 제어 메시지는 XML 형식으로 다음과 같은 모습을 한다.

```
〈ControlComponentCommand〉
  〈Receiver〉 ControlCompnent Name 〈/Receiver〉
  〈Type〉 Command Type 〈/Type〉
  〈Parameters〉 Parameter 〈/Parameters〉
〈/ControlComponentCommand〉
```

제어 컴포넌트 매니저가 제어 요청 메시지를 받으면 해당 제어 컴포넌트에게 제어 요청을 전달한다. 단 LOAD 요청은 제어 컴포넌트 매니저가 직접 처리한다. LOAD 요청을 받은 제어 컴포넌트 매니저는 요청 메시지에 포함된 제어 컴포넌트의 물리적인 위치 정보를 통해 해당 제어 컴포넌트를 메모리에 적재한다. 메모리에 적재된 제어 컴포넌트의 최소 실행 상태는 STOPPED이다. 나머지 명령들은 제어 컴포넌트에게 전달되며 현재 실행 상태에서 수락할 수 있는 요청인지 검사한다. START 요청과 RESUME 요청은 즉시 처리되며 SUSPEND 요청과 STOP 요청을 제어 컴포넌트가 실행 중 tmCheckPoint()를 호출하는 시점에서 처리된다.

제어 컴포넌트의 실행 상태는 변경과 동시에 제어 컴포넌트 매니저를 거쳐 작업 관리 커널에 보고된다. 이 역시 XML 형식이며 다음과 같은 모습을 한다.

```
〈ControlComponentStatusReport〉
  〈Sender〉 ControlComponent Name 〈/Sender〉
  〈Status〉 ControlComponent Status 〈/Status〉
〈/ControlComponentStatusReport〉
```

변경된 제어 컴포넌트의 실행 상태는 제어 컴포넌트 매니저와 작업 관리 커널에 반영되며, 작업 관리 커널

은 변경 사항을 작업 관리기에게 보고한다. 작업 관리기는 제어 컴포넌트의 실행 상태를 인지하고 제어 로직을 수행하는 동안 판단의 근거로 활용한다.

제어 컴포넌트에게 정보를 제공하는 방법도 제어 명령과 동일한 형식으로 정의하며 그것과 동일한 형식의 XML 메시지로 변환하여 전달한다.

INFORM	제어 컴포넌트에게 정보 전달

제어 컴포넌트에게 정보를 전달하는 것은 직접적으로 제어 컴포넌트의 실행 상태를 변경시키지 않는다. 전달된 정보를 통해 수행 중인 작업의 내용을 변경하거나 수행을 종료하는 것은 제어 컴포넌트의 선택이다.

7. 통합 관리기의 구현

지금까지 설명한 통합 관리 구조에 기초하여 통합 관리기를 구현하였다. 작업 관리기와 작업 관리 커널, 제어 컴포넌트 매니저는 Java 언어로 구현하였으며 Java 언어의 특성상 통합 관리기는 Java 가상 머신을 구동시킬 수 있는 다양한 플랫폼에서 동작할 수 있다. 더불어 메시지 기반의 통신과 분산 환경의 지원은 제어 컴포넌트의 배포를 자유롭게 할 뿐만 아니라 작업 관리기의 위치 또한 자유롭게 한다.

작업 관리기는 BDI 기반의 지능형 에이전트 시스템인 JAM 에이전트 시스템을 적극 활용하였고 작업 관리기의 상황을 모니터링하기 위한 모니터 도구를 제공한다(그림 5).

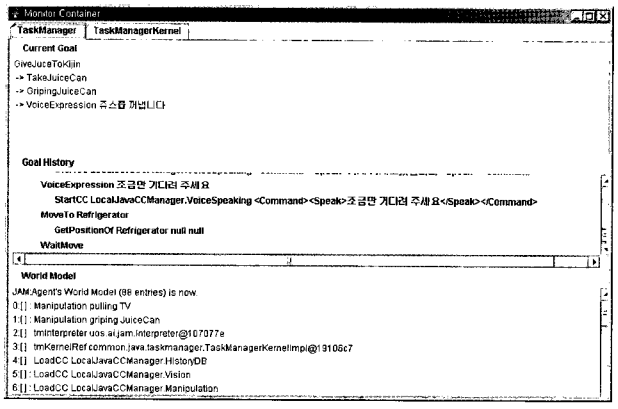

그림 5 작업 관리기 모니터

작업 관리 커널은 모든 제어 컴포넌트의 실행 상태가 종합되고 작업 관리기로부터의 제어 요청과 제어 컴포넌트로부터의 요청 수락, 제공된 정보를 연계하므로 이러한 상황을 모니터링할 수 있는 모니터 도구를 제공한다(그림 6).

제어 컴포넌트를 쉽게 구현할 수 있도록 제어 컴포넌트의 템플릿(추상 클래스)을 제공하며 Java와 C++

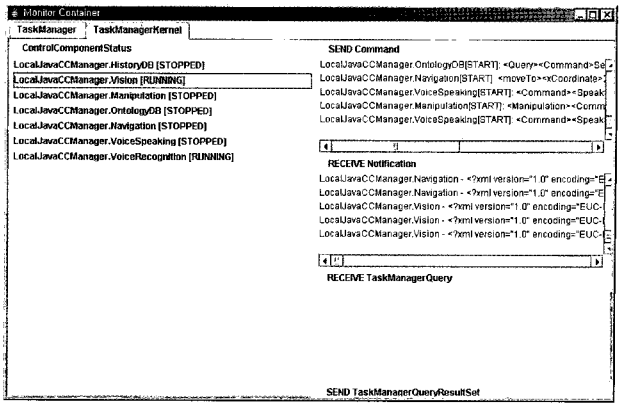

그림 6 작업 관리 커널 모니터

언어로 구현된 템플릿을 동시에 제공한다. 대부분의 로봇 기술이 Java와 C++ 언어로 구현되어 있으므로 두 언어로 구현된 제어 컴포넌트 템플릿을 통해 기존의 기술을 쉽게 통합할 수 있다.

8. 실 험

통합 관리기를 실험하기위해 간단한 로봇 시뮬레이션 환경을 구성했다. GUI 기반의 시뮬레이터(그림 7)에는 노인과 로봇, 냉장고, TV 등이 존재하며 노인의 행위를 GUI 인터페이스를 통해 제어할 수 있고 로봇을 제어하기 위한 센서와 휠 등의 가상 디바이스를 제공한다.

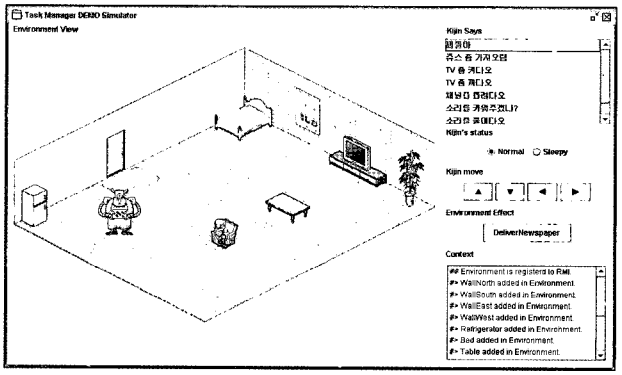

그림 7 시뮬레이터

시뮬레이터가 제공하는 로봇의 가상 디바이스는 리액티브 계층에 속한다. 이러한 가상 디바이스를 이용해 주행, 음성 발화, 물체 인식, 조작 등의 기능을 수행하기 위한 소프트웨어 컴포넌트를 제작하였으며 이것들은 시퀀싱 계층에 속한다.

시퀀싱 계층의 소프트웨어 컴포넌트는 딜리버레이트 계층과 상호 작용하기 위해 제어 컴포넌트화 해야 한다. 여기서는 각각의 소프트웨어 컴포넌트가 개별적으로 행위 조작의 대상이 되므로 각각을 모두 제어 컴포넌트화 하였다.

작업 관리기에는 간단한 심부름을 수행할 수 있는 실행 계획들이 미리 정의되어 있다. 실행 계획은 주어진 심부름을 수행할 수 있도록 제어 컴포넌트를 조작하는 절차를 서술하거나, 주어진 목적을 세부 목적으로 잘게 나눈다. 다음은 주스 심부름을 수행하는 실행 계획이다.

```
PLAN:
{
   NAME: "GiveJuceToKijin"
   GOAL: ACHIEVE GiveJuceToKijin;
   BODY:
      PERFORM VoiceExpression "주스를 가져다 드리겠습니다.";
      PERFORM TakeJuiceCan;
      PERFORM GiveJuiceCanTo "Kijin";
}
```

주스 심부름을 수행하는 실행 계획의 내용은 세 개의 세부 목적을 차례로 달성하는 것이다. (1)음성 발화를 하고 (2)주스를 획득하여 (3)노인에게 가져다준다. 작업 관리기의 JAM 에이전트 시스템은 세부 목적을 달성할 수 있는 실행 계획을 검색하고 검색된 실행 계획 중에서 현재 상황에 가장 적절한 수행 계획을 선택하여 실행한다.

두 번째 세부 목적인 '주스 획득'을 수행할 수 있는 실행 계획 중 하나는 다음과 같다.

```
PLAN:
{
   NAME: "TakeJuiceCan"
   GOAL: ACHIEVE TakeJuiceCan;
   PRECONDITION:
      FACT HAS "JuiceCan" "NO";
   BODY:
      PERFORM VoiceExpression "쥬스가 냉장고에 있으니";
      PERFORM VoiceExpression "가서 가져오겠습니다";
      PERFORM VoiceExpression "조금만 기다려 주세요";
      PERFORM MoveTo "Refrigerator";
      PERFORM WaitMove;
      PERFORM GripingJuiceCan;
}
```

세부 목적에 의해 수행되는 실행 계획은 또다시 세부 계획을 생성한다. 이렇게 실행 계획이 주어진 문제를 작은 문제로 나누고 다시 나눠진 작은 문제들을 해결하기 위해 실행 계획을 검색하는 과정을 통해 작업 관리기는 보다 다양한 행위의 조합을 만들어낼 수 있다. 마지막으로 실제로 제어 컴포넌트를 조작하는 실행 계획의 예로 이것은 음성 발화 제어 컴포넌트를 조작한다.

```
PLAN:
{
   NAME: "VoiceExpression"
   GOAL: ACHIEVE VoiceExpression $EXPRESSION;
   BODY:
      PERFORM StartCC "LocalJavaCCManager.VoiceSpeaking"
         (+ "<Command><Speak>" $EXPRESSION
            "</Speak></Command>" );
}
```

```
PLAN:
{
   NAME: "ControlComponent START"
   GOAL: ACHIEVE StartCC $NAME $PARAM;
   PRECONDITION:
      RETRIEVE tmKernelRef $KERNEL;
      FACT controlComponentStatus $NAME "STOPPED";
   BODY:
      ATOMIC {
         RETRACT controlComponentStatus $NAME;
         EXECUTE uos.ai.tmdemo.primitive.ActionControlCC.execute
                  $KERNEL $NAME "START" $PARAM;
         WAIT controlComponentStatus $NAME;
      };
}
```

작업 관리기와 작업 관리 커널이 제공하는 모니터 도구를 이용해 현재 로봇 시스템 내부에서 이루어지는 진행 현황을 파악할 수 있다.

다음은 실제 로봇 환경에서 실험한 결과이다(그림 8). 로봇의 주된 목적은 손님에게 음료수를 제공하는 것으로 앞서 설명한 3-계층 아키텍처로 구성되어 있다. 로봇의 시퀀싱 계층에는 주행, 조작, 인식 등의 다양한 로봇 기술들이 존재하며 딜리버레이트 계층과 상호 작용하는 기술들은 제어 컴포넌트화하였다.

작업 관리기에는 주된 목적인 주스 서비스 외에도 주문 접수, 알고 있는 손님의 인식, 손님과의 대화 등 다양한 실행 계획이 미리 정의되어 있다. 로봇은 손님을 인식하거나 대화를 통해 새로운 목적을 생성해내고 적절한 실행 계획을 검색하여 수행한다.

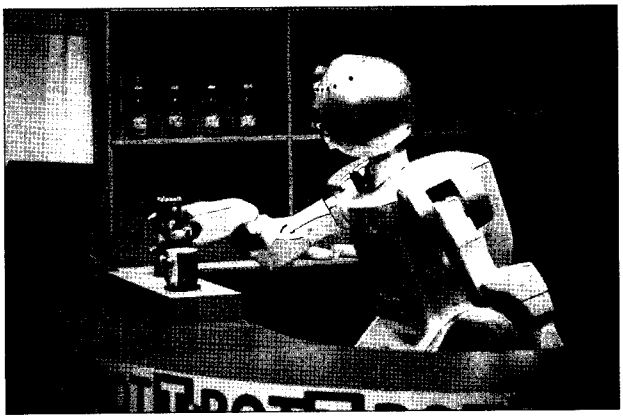

그림 8 카페 로봇

9. 결론 및 향후 계획

통합 관리기는 이미 존재하는 로봇 기술들을 쉽게 통합할 수 있는 방법을 제공한다. 각 기술은 작업 관리기와 직접 연결되지 않고 제어 컴포넌트를 통해 연결되므로 기술의 유지보수 작업에 대한 파급 효과를 최소화할 수 있으며 새로운 기능의 추가를 용이하게 한다. 또한 작업 관리기와 제어 컴포넌트간의 인터페이스가 매우 단순하고 함수 호출이 아닌 메시지를 통한 요

청과 수락을 통해 제어되므로 다양한 플랫폼을 지원할 수 있으며 로봇과 직접적인 연관이 없는 다른 기술들도 쉽게 통합할 수 있다.

통합 관리 구조가 분산 환경을 지원함에 따라 제어 컴포넌트를 적절히 배포함으로써 로봇의 자원을 효율적으로 사용할 수 있다. 제어 컴포넌트의 배포 상황은 제어 컴포넌트 배포 설명서로 분리되어 있으므로 제어 컴포넌트의 배포 상황을 간단히 수정할 수 있다.

제어 컴포넌트의 추상화된 제어와 제어 컴포넌트 배포 설명서를 이용한 배포를 통해 새로운 로봇 기술은 통합 관리 구조에 쉽게 통합될 수 있다. 새로운 기술은 적절한 제어 컴포넌트의 제어 하에 들어가거나 그것을 직접 제어 컴포넌트화하고 제어 컴포넌트 배포 설명서에 배포 상황을 서술하는 것만으로 로봇 시스템에 통합된다. 작업 관리기는 새로운 제어 컴포넌트를 사용하기위해 필요한 부가 정보는 알아야 하지만 그것을 제어하는 방법은 일관성을 유지하므로 쉽게 사용할 수 있다. 작업 관리기에 이미 존재하는 실행 계획들은 제어 컴포넌트의 추가에 대해 영향을 받지 않으므로 기존의 동작은 여전히 유효하다. 새 제어 컴포넌트를 활용하기 위해서는 기존의 실행 계획을 수정하는 것이 아니라 그것을 활용하는 새로운 실행 계획을 추가해야 한다. 이것은 기존의 동작을 여전히 유효하게 하며 새로운 실행 계획과의 조합을 통해 좀 더 다양한 행위의 취할 수 있게 한다.

본 논문에서는 시스템 통합 중에서도 제어에 관련된 통합을 다루었다. 로봇 시스템이 보다 효과적으로 통합되기 위해서는 제어뿐만 아니라 지식에 대한 통합이 필요하다. 작업 관리기와 제어 컴포넌트간의 지식의 공유, 생산, 수정, 검색 등에 대한 지원과 지식의 효과적인 관리가 통합 관리 구조 차원에서 지원되어야 할 것이다. 또한 생산된 지식은 확장될 수 있어야하며 성장할 수 있어야 한다[1, 4]. 지식의 통합을 위해 온톨로지 기반의 모델[7]을 적용할 수 있으며 지식 포털 서비스와의 연동을 꾀할 수도 있을 것이다. 통합 관리되는 지식을 단순한 정보의 집합에 그치게 하지 않고 학습, 적응, 성장 기능을 접목하여 보다 의미 있는 정보로 발전시키고 이것을 적극 활용하면 로봇의 행위를 더욱 지능적으로 향상시킬 수 있을 것이다.

작업 관리기에 의한 제어 컴포넌트로의 제어 요청은 서비스 제공자로의 서비스 요청으로 그 의미를 확장시킬 수 있으며 이와 함께 분산 환경의 지원과 메시지 기반의 통신은 로봇을 서비스 중심의 유비쿼터스 환경으로 확장시키기 용이하게 한다. 이를 통해 로봇 기능의 범위를 웹 서비스 영역으로 확장시킬 수 있으며 홈 네트워크와의 연동을 꾀할 수 있을 것이다.

통합 관리 구조는 멀티 로봇 환경으로 확대 응용할 수 있다. 지능형 에이전트로 구현된 다수의 작업 관리기가 ACL(Agent Communication Language) 메시지를 통해 서로 협동하고 공조하여 공동의 문제를 해결하기 위한 작업을 수행할 수 있을 것이다[2]. 유비쿼터스 환경으로의 확장[9]과 멀티 로봇 환경의 지원은 커뮤니티 컴퓨팅 개념을 도입할 수 있게 하며 이를 통해 다수의 로봇이 서로 협동하는 체계를 구축할 수 있을 것이다.

참고문헌

[1] Dzbor, M., Paralic, J., and Paralic, M., "knowledge management in a distribute organization," Proceedings of BASYS '2000 - 4th IEEE/IFIP International Conference on Information Technology for Balanced Automation Systems in Manufacturing, Kluwer Academic Publishers, London, ISBN 0-7923-7958-6, pp.339-348, 2000.

[2] FIPA, "FIPA Organization Home Page," available at http://fipa.org, 1997.

[3] Jaeho Lee, Edmund H. Durfee, "Structured circuit semantics for reactive plan execution systems," Proceedings of the twelfth national conference on Artificial intelligence(Vol.2), pp.1232-1237, October, 1994.

[4] Jarrar, Y., Schiuma, G., and Zairi, M., "Defining organizational knowledge: a best practice perspective," Proceedings of VI International Conference on "Quality Innovation Knowledge," pp.486-496, 2002.

[5] Marcus Huber, "JAM, A BDI-theoretic Mobile Agent," Proceedings of the Third International Conference on Autonomous Agent(Agent-99), May, 1999.

[6] Michael Beetz, "Autonomous Agent and Multi-Agent System," Volume 4, Issue 1-2, pp.25-55, March-June 2001.

[7] Mike Dean, et al, "OWL Web Ontology Language Reference," http://www.w3.org/TR/owl-ref/

[8] Selahattin Kuru, Ferda Bek, "Goal-driven blackboard control architecture based on extending partially complete general goal

trees," Volume 3, Number 4, December 1990.
[9] Weiser, M., "Some computer science issue in ubiquitous computing," Communications of the ACM, Volume 36, Issue 7, pp.75-84, 1993.

곽 별 샘

2006 서울시립대학교 전자전기컴퓨터공학부 (학사)
현 재 서울시립대학교 전자전기컴퓨터공학부 석사과정
관심분야 : 지능형 에이전트 시스템, 멀티 에이전트 시스템
E-mail : semix2@naver.com

이 재 호

1985 서울대학교 계산통계학과(학사)
1987 서울대학교 계산통계학과(석사)
1997 University of Michigan(박사)
1998~현재 서울시립대학교 전자전기컴퓨터공학부 부교수
관심분야 : 에이전트 기반 시스템, 인공지능, 온톨로지 기반 시스템, 지능형 로봇 시스템
E-mail : jaeho@uos.ac.kr