상태 인식에 따른 자율 주행 에이전트 시스템

정슬기*, 이태경**
*,**동국대학교 전자계산학과
e-mail: lovehard@dongguk.ac.kr*, tklee@dongguk.ac.kr**

Autonomous Car-Driving Agent System Based on State Recognition

Suelki Jung*, Taekyung Lee**
*,**Department of Computer Science, Dongguk University

요 약

본 논문은 에이전트빌더를 통해 자동차 자율 운행을 위한 자동 주행 에이전트, 위치 파악 에이전트, 상태 점검 에이전트가 협동하는 멀티 에이전트를 구현하였다. 멀티에이전트의 협동 및 제어를 위해서 에이전트빌더에 내에 메타 제어기의 구성을 보여 주었으며, 현실에서의 자동차 주행 시 일반적으로 생기는 환경변수를 고려하고 일정 지역을 통하여 자율 주행에 대한 가능성을 찾았다.

.

1. 서 론

에이전트 시스템은 감각기관, 환경, 작용기 세 가지로 구성되어 있는데 이 시스템은 자율적인 프로세스라는 것과 사용자가 원하는 작업을 자동적으로 해결해 준다는 점을 특징으로 한다. 또한 지식이 저장된 데이터베이스를 이용하고, 자신의 추론 방법을 통해 다른 에이전트와 상호 작용을 할 수 있게 해주며, 경험을 통한 학습기능 및 목적 지향적 능력을 갖추고 있다.

멀티에이전트는 단일 에이전트가 해결하지 못하는 복잡한 문제의 해결을 위해서 다수의 에이전트를 이용하여 설계한다. 또한 서로 다른 시각과 해결방식을 가지고 있는 에이전트로 구성된 문제를 표현하기에 적합하다[1].

본 논문에서 구현하는 시스템은 주행 처리, 상태 점검, 위치 파악 에이전트로 구분된다. 에이전트 시스템은 에이전트빌더 Pro 1.4를 이용하여 구현하였고 인터페이스는 자바 프레임을 사용하였다.

2. 관련 연구 및 분석

2.1 에이전트 시스템의 연구 동향

현재 에이전트 이론에 대한 연구에서는 Z언어를 이용하여 에이전트의 각 특성을 에이전트 환경과 행동으로 연계시켜 정형화한다. 또한 멀티 에이전트 시스템을 선언적 표현방법을 사용하여 정의하고 있다. 에이전트 구조에 관한 연구에서는 정형화한 특성들을 구현하기 위한 에이전트 구성요소와 제어 및 통신 프로토콜 등을 대상으로 한다. 에이전트 언어에 관한 연구는 에이전트를 개발하기 위해 에이전트 특성을 고려한 특정 프로그래밍 언어를 개발하는데 초점을 맞추고 있다. 에이전트 이론에서 제시된 인지요소를 직접 이용한 에이전트 언어는 크게 AOP, PLACA, ABLE와 같은 에이전트 기술언어와 Java, Telescript와 같은 에이전트 개발 언어로 구분되고 있다[2].

에이전트 응용은 앞서 언급한 이론과 구조 등을 기반으로 하여 실제 사용할 수 있는 응용 소프트웨어를 개발하는 것이다. 인터넷 정보검색, 온라인 쇼핑,

메시징, 네트워크 관리 등 에이전트를 필요로 하는 분야는 매우 많다.

2.2 에이전트 시스템의 구성 및 설계

멀티에이전트 시스템은 여러 응용 에이전트 외에 조정 에이전트라는 중재자를 통해 메시지를 전달하고 각 에이전트의 제어를 수행한다[3]. 이 경우 모든 응용 에이전트의 통신 메시지는 조정 에이전트를 통해 다른 에이전트로 전달된다. 각 에이전트가 어떤 처리 능력이 있는지에 대한 정보는 응용 에이전트 생성 시에 조정 에이전트로 등록되며, 조정 에이전트는 이를 바탕으로 통신 메시지를 해당 응용 에이전트에게 보낸다. 조정 에이전트는 응용 에이전트의 수행 능력을 통해 에이전트의 실체를 알 수 있는 에이전트 네임 서버의 역할을 수행하는 것이다.

멀티에이전트에서 발생하는 중요한 문제인 이형질성을 해결하기 위해서는 이형질의 응용 에이전트간의 통신이 요구된다. 대표적으로 에이전트와 에이전트의 통신은 KQML메시지를 통해 이루어지며 이외에 프로토콜을 이용하여 메시지 교환을 하는 방식, CORBA를 통해 이루어지는 방식이 있다.

3. 상태 인식에 따른 자율 주행 에이전트 시스템

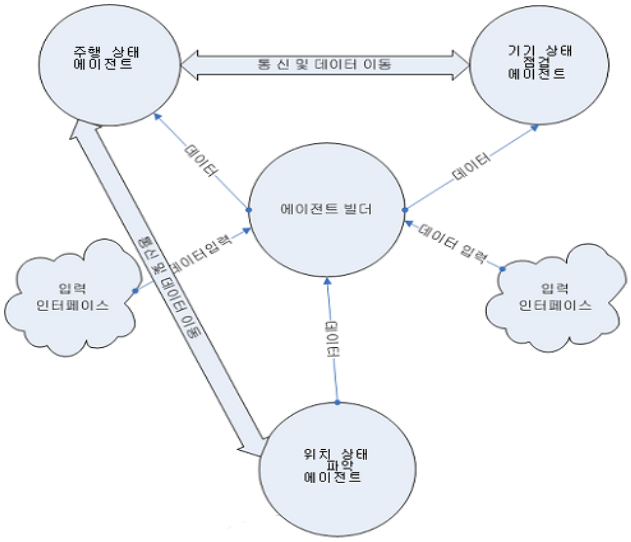
3.1 시스템 전체구성

상태 인식에 따른 자율적인 주행을 위한 멀티에이전트 구조에서는 에이전트 빌더가 중심에 존재한다. 그리고 이 에이전트 빌더가 자동 주행 에이전트, 상태 점검 에이전트, 위치 파악 에이전트를 관리한다. 그림 1은 상태인식에 의한 자동차 자율 운행 멀티에이전트의 구성도를 보여준다.

3개의 에이전트는 에이전트빌더 안에 존재하며, 에이전트 빌더는 각 인터페이스로부터 오는 정보를 통해 어떤 에이전트에 전달해야 하는 지를 판단한다. 에이전트들 간의 통신이 필요할 때에도 에이전트 빌더는 송신자와 수신자가 정확한지를 판단하여 전달한다. 3개의 에이전트를 조정하는 역할은 에이전트빌더의 메타 지식들을 이용하여 수행된다.

3.2 주행 상태 에이전트의 구성

자동 주행 에이전트는 자동차의 자율 운행에 있어서 핵심적인 에이전트이다. 이 에이전트는 탑승자로부터 목적지를 입력 받고 이를 위치 파악 에이전트에 전달한다. 주행상태 에이전트는 기후상태 파악과 처리, 주변 차량 파악, 차량 문제 처리, 목적지 파악과 처리, 주행처리 모듈 등 총 5가지의 모듈로 구성된다.



[그림 1] 자율 주행 에이전트 시스템 전체구성도

3.3 기기 상태 점검 에이전트의 구성

자동차에 다양한 부품상태를 알려주는 센서가 존재하지 않기 때문에 가상의 인터페이스를 통하여 에이전트에게 센서를 대신하여 정보를 전달하도록 한다. 기기 상태 점검 에이전트는 차체부, 엔진부, 전기장치의 3개의 모듈로 구성된다.

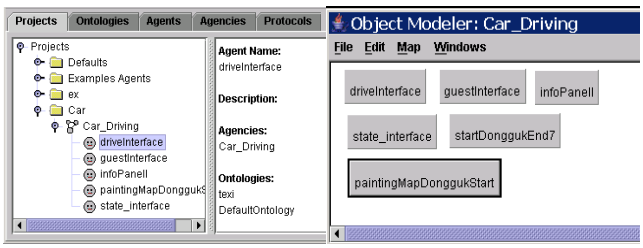
3.4 위치 상태 파악 에이전트의 구성

위치 파악 에이전트는 자동차의 위치를 파악하고 목적지까지의 경로를 탐색하여 알려주는 역할을 한다. 현재 위치 파악, 최종 목적지 파악, 중간 경유지 파악, 이동 좌표 송출, 신호 상태 파악의 모듈로 구성된다.

4. 구현 및 실험분석

4.1 메타 제어기

실험에 사용된 운영체제는 윈도우 2003 서버이고, 입력 인터베이스는 java로 구현하였으며, java jdk 1.4 버전으로 컴파일을 하였다. 에이전트들을 구성한 툴로는 에이전트빌더Pro 1.4가 사용되었다. 그림 2는 주행 상태, 기기 상태점검, 위치 상태파악 에이전트를 관리 및 제어를 해결하기 위하여 메타 제어기에 의해 처리되는 정보의 구조를 보여준 것이다[4][5].

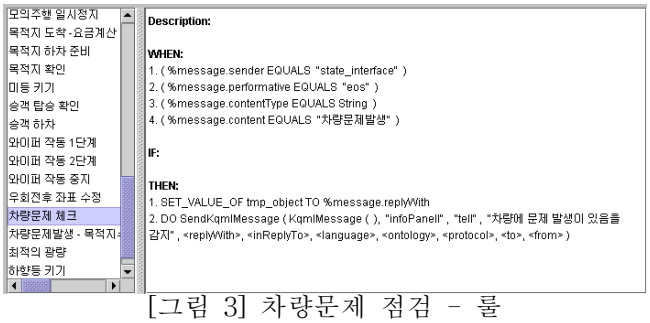


[그림 2] 메타 제어기의 구성

4.2 주행 상태 에이전트 구현

(1) 차량문제 점검 처리

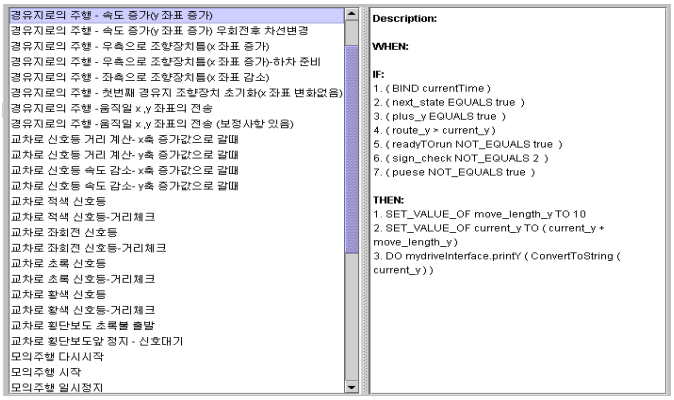
차량의 상태점검 에이전트로부터 문제가 있음을 알리는 메시지를 감지할 경우 인포패널에 문제를 감지함을 알린다.



[그림 3] 차량문제 점검 - 룰

(2) 주행처리

위치파악 에이전트로부터 받은 경유지의 좌표를 받게 된다. 이에 따라서 현재 위치와 목표 위치를 비교하여 해당되는 룰을 발생시키게 되고 해당하는 룰에서 차량이 움직일 좌표를 결정하게 된다. 이후에 움직인 좌표를 위치파악 에이전트로 전송한다. 속도의 증감 처리, 교차로의 신호 상태를 확인하여 거리계산 후 횡단보도 정지선 앞에서의 정지, 출발. 우회전의 처리 등의 룰은 각 조건을 비교하여 처리된다.



[그림 4] 속도 증가 - 룰

4.3 기기 상태 점검 에이전트

자동차의 상태를 점검하기 위해서는 실제의 센서가 없기 때문에 센서 역할을 대신하는 인터페이스를 구현한다. 상태 점검 인터페이스는 차체부, 엔진부, 전기장치의 3개의 모듈로 나누어 구현되었다.

코드 1은 엔진부 모듈로서 적정 값이 정해져 있다. 텍스트 필드에 값을 입력하고 라디오 버튼을 클릭하면 적정 값과 비교하여 이벤트가 발생한다. 라디오&텍스트 필드 버튼이 클릭되면 즉, 이벤트가 발생

```
if(jRadioButton22.isSelected()&& r22==0)  {
    textArea1.append(R22_LABEL +
    jTextField4.getText() +  " 이상있음\n");
    message.setPerformative("achieve");
    message.setContent(R22_LABEL);
    message.setReplyWith(jTextField4.getText());
    r22 =1; }
```

[코드 1] 엔진부 모듈 인터페이스

하면 체크되는 카운트를 세고 텍스트 박스에 이벤트 발생 시 나타나는 메시지를 출력한다. 그리고 그 메시지 형태로는 'archieve' 이고, R22_LABEL을 전달한다. 메시지 형태와 R22_LABEL은 에이전트 빌더로 구현할 때 에이전트 룰의 원형으로 쓰인다. 그리고 정수형 변수 r22는 초기화를 위해 쓰인다.

```
jRadioButton1.setActionCommand(R1_LABEL);
jRadioButton1.addActionListener(this);
if (jRadioButton1.isSelected() && r1== 0) {
   count++;
   textArea1.append(R1_LABEL
     + " 이상있음\n");
   message.setPerformative("achieve");
   message.setContent(R1_LABEL);
   r1 = 1;  }
```

[코드 2] 차체부 전기장치 모듈 인터페이스

코드 2는 차체부 전기장치 모듈로서 라디오 버튼이 클릭되면 즉, 이벤트가 발생하면 체크되는 카운트를 세고 텍스트 박스에 이벤트 발생 시 나타나는 메시지를 출력한다.

4.4 위치파악 에이전트

(1) 기본 그래픽 처리 부분

화면에 나타나는 모든 차선과 자동차, 신호등은 자바 언어를 이용하여 나타내며, 전체 화면크기는 500, 500의 크기로 하고, setColor()로 색을 표현하며 drawLine()로 선을 그려주었다. 위치는 x, y를 이용하여 좌표로 나타내 준다.

(2) 현재 위치 파악

에이전트빌더에서 함수를 호출하여 주행 처리, 즉 에이전트가 소지하고 있는 현재 위치 좌표정보를 싱크 시키는 부분이다.

```
public void position(String p_x, String p_y){
  x = (Integer.parseInt(p_x) * -1);
  y = Integer.parseInt(p_y); }
```

[코드 3] 현재 위치 호출

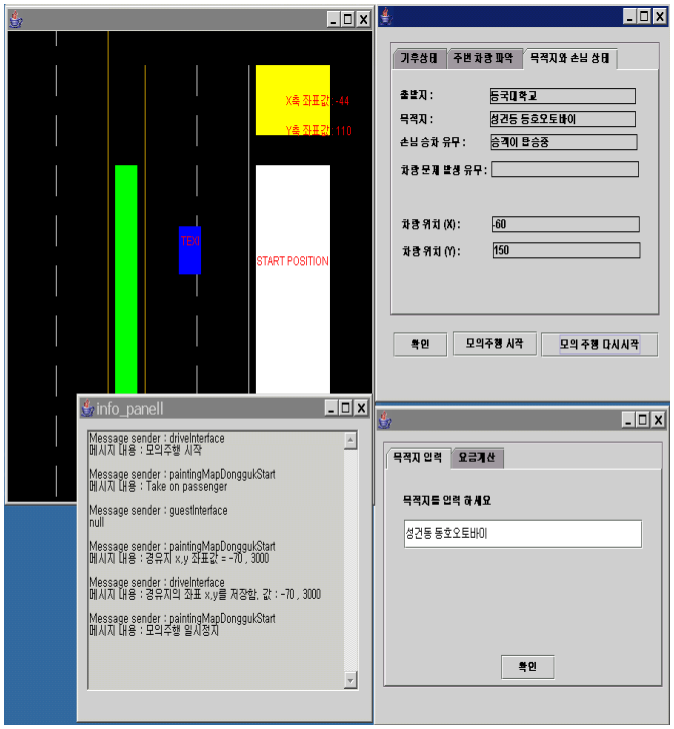
(3) 신호 상태 파악

신호등은 푸른등, 황색등, 적색등, 좌회전 표시등의 네 가지로 표현하였다. 0은 적색등, 1은 좌회전표시등, 2는 초록등, 3은 황색등을 나타내는 것이다. 신호등은 0→3→2→1의 순서로 진행된다.

```
page.setColor(Color.gray);
page.fillRect(MID-150+x, TOP -3800+y, 40, 30);
if (light == 0){
  page.setColor(Color.red);
```

[코드 4] 신호등을 나타내는 부분(적색등)

4.5 실험 결과

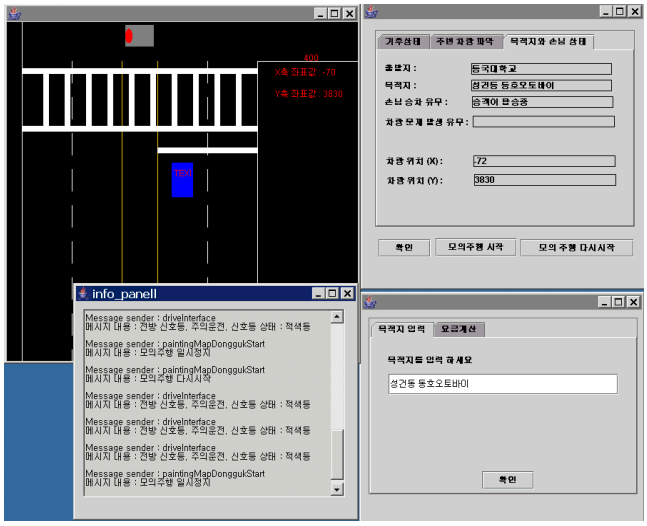


[그림 5] 실행 화면 1 - 목적지 입력

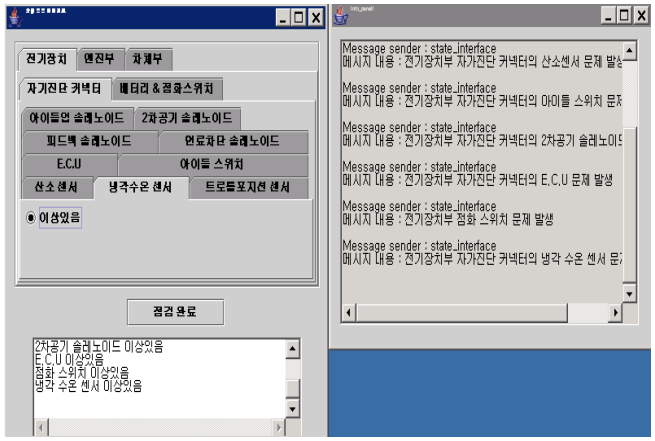
그림 5처럼 목적지의 입력이 완료되면 출발지와 목적지가 설정된다. 이후 경유지를 전송받아 경유지로의 주행이 시작된다. 그림 6은 횡단보도 앞에서 정지한 후 초록색 신호등을 대기 하고 있는 상태이다. 그림 7은 에이전트빌더로 전기장치를 점검하는 상태이다.

5. 결론

본 논문은 에이전트빌더를 통한 자동차 자율 운행을 위하여 자동 주행 에이전트, 위치 파악 에이전트,



[그림 6] 실행 화면 2 - 신호등 처리



[그림 7] 실행화면 3 - 전기장치

상태 점검 에이전트가 복합된 멀티에이전트를 구현하였다. 실제 현실에서의 자동차 주행과 동일하지는 않지만 가장 일반적으로 생기는 환경변수를 고려하고 일정 지역을 통하여 자율 주행에 대한 가능성을 찾았다.

참고문헌

[1] Walter Brenner, Intelligent Software Agents, Springer, 1998

[2] Lin Padgham., Michael Winikoff, Developing Intelligent Agent Systems, Wiley, 2004

[3] Peter Stone, Multiagent Systems: a Survey from a Machine Learning Perspective, Carnegie Mellon University, 1997

[4] Acronymics, Inc., Agent Builder User's Guide. 2004

[5] Acronymics, Inc., Agent Builder Reference Manual. 2004