# NPC 행동 제어를 위한 유한상태기계와 추론 엔진의 하이브리드 구조†

NPC Control by Hybrid Architecture of Finite State Machine and Inference Engine

조동현, Donghyun Cho*, 오성진, Sungjin Oh*, 성미영, Meeyoung Sung**, 전경구, Kyungkoo Jun*
*인천대학교 멀티미디어시스템공학과, **인천대학교 컴퓨터공학과

**요약** 게임이나 가상환경에서 오락성과 실감성을 증진시키는 여러 가지 방법들 가운데 지능적인 Non-Player Character (NPC)들의 존재는 중요하다. 컴퓨터 그래픽과 관련 하드웨어 플랫폼 기술의 발전으로 인해 사용자들은 이제 시각적인 만족을 넘어서서, NPC들이 보다 지능적으로 행동하면서 오락적인 만족감과 동시에 보다 향상된 실감성을 제공하기를 원한다. 하지만, 유한상태기계 (Finite State Machine, FSM)를 기반으로 하는 NPC 구현의 한계와 어려움으로 인해 이러한 사용자들의 요구사항을 만족시키는 것은 어렵다. 본 논문에서는 FSM과 추론 엔진 (Inference Engine)을 결합한 새로운 NPC 행동제어 구조를 제안한다. 또한 제안된 구조의 가능성을 시연하기 위해 실제로 동작하는 데모를 소개한다. 이러한 FSM과 추론 엔진의 하이브리드 구조는 FSM이 제공하는 NPC 반응의 실시간성을 보장하는 동시에 추론 엔진이 제공할 수 있는 보다 지능적이고 계획적인 NPC들의 행동을 만들어 낼 수 있다는 장점이 있다.

***핵심어: Non-Player Character, Artificial Intelligence***

## 1. 서론

게임이나 가상환경에서 오락성과 실감성을 증진시키는 여러 가지 방법들 가운데 지능적인 Non-Player Character(NPC)의 역할은 중요하다[1][3]. 하지만 게임이나 가상환경이 복잡해짐에 따라 좀 더 지능적으로 행동하는 NPC가 필요해졌다. 게임의 예를 들자면, NPC들이 사용자 캐릭터와 협조하여 임무를 완성하는 게임의 경우, NPC들이 사용자 캐릭터가 수행하는 임무를 돕는 것은 훨씬 지능적이고 때에 따라서는 계획단계까지도 포함하는 복잡한 일이다. 또한 게임이나 가상환경은 여러 명의 사용자가 동시에 네트워크로 연결되어 있는 경우가 많아, NPC 행동을 결정하는 데 있어, 다수 사용자 행동을 모두 고려해야 함에 따라 그 복잡성의 정도가 더 심해진다.

지금까지 NPC 구현은 주로 유한상태기계 (Finite State Machine, FSM) 기법에 기반 하였다. FSM은 유한 개의 상태와 상태간 전이를 정의하는 조건으로 구성되어 있다. 이러한 FSM 기반 기법은 짧은 제한시간 내에 NPC 행동을 제어할 수 있으며, 또한 그 구현이 간단하여 매우 효율적이라는 장점을 가지고 있다.

그러나 FSM은 상태의 수와 상태전이가 정해져 있어 이것에 기반한 NPC가 지능적인 행동을 보이게 만들기 어렵다. 왜냐하면, 보다 지능적인 NPC를 만들기 위해서는 FSM이 더 많은 수의 상태와 상태전이를 가지게 되므로 구현이 매우 복잡해지기 때문이다. 게다가, FSM은 주어진 상황에 대한 지식을 이용하여 새로운 지식을 추론해내는 능력을 가지지 못해 지능적 NPC 제어에는 한계가 있다.

컴퓨터 그래픽과 관련 하드웨어 플랫폼 기술의 발전으로 인해 사용자들은 이제 시각적인 만족을 넘어서, NPC들이 보다 지능적으로 행동하면서 오락적인 만족감과 동시에 보다 향상된 실감성을 제공하기를 원하지만 기존의 FSM기반 NPC 구현은 이러한 사용자 요구사항을 만족시키는 데는 한계가 있다.

본 논문에서는 이러한 FSM에 기반한 NPC 구현의 단점을 보완하여 보다 지능적인 NPC 행동제어 방식을 제공하고자 FSM과 추론엔진 (Inference Engine)을 결합한 형태의 NPC 제어구조를 제안하고, 이 구조의 가능성을 보이고자 제안구조에 따라 구현된 데모를 소개한다.

† 본 연구는 산업자원부 지방기술혁신사업(RTI05-03-01) 지원으로 수행되었음

## 2. NPC 제어를 위한 유한상태기계와 추론엔진의 하이브리드 혼합구조

기존 대부분의 NPC 동작 제어는 유한상태기계 (Finite State Machine, FSM) 기법에 기반을 두어왔다. 유한상태기계는 해당 시스템에서 있을 수 있는 유한개의 상태를 가지고 입력에 따라 현재의 상태에서 다른 상태로 전환하거나 출력 혹은 액션이 일어나게 하는 장치 또는 그런 장치를 나타낸 모델이다.

NPC 제어에서의 FSM 방식은 가질 수 있는 상태들을 미리 정의하여 두고 발생한 사건 등을 기준으로 상태 전이를 일으킨다. 각 상태마다 NPC의 동작을 나타내는 제어 방식 등이 서술되어 있기 때문에 NPC가 행할 수 있는 동작의 다양성은 상태의 개수에 따라 결정된다. 하지만 이러한 FSM 방식의 NPC 제어는 미리 정의되어 있는 상태와 그에 따른 몇 가지의 분기에 의해서만 이뤄지기 때문에 현재 NPC 상태와 발생하는 사건을 가지고 간단히 NPC 행동을 예측할 수 있다. 예를 들어, 가상공간에서 어머니와 아이를 나타내는 두 NPC들이 있다고 하자. 이때, 가상공간을 다니던 아이 NPC가 놀이터에 마음이 끌려 어머니 NPC 곁을 떠난다. 이 경우, FSM만으로 구현했다면, 어머니 NPC는 사라진 아이 NPC를 찾기 위해 전체 공간을 순차적으로 탐색하는 행동만을 보이게 된다.

본 논문에서는 FSM 방식만을 이용한 NPC 제어 방식의 단순함을 극복하고, 보다 지능적인 NPC 행동제어 방식을 제공하고자 FSM과 추론엔진을 결합한 형태의 NPC 제어구조를 제안한다.

추론엔진이란 정보에 대해 추론하고 결론을 형식화하는 방법론을 제공하는 것으로 간단하게 말해 fact와 rule을 조합하여 결론을 도출하는 방법을 제공한다. 추론이란 이미 알고 있는 명제를 기초로 하여 새로운 명제를 유도하는 과정을 말하는데 추론엔진에서는 전제(premise)와 결론(conclusion)간의 논리적인 관계를 다룬다. 여기서 전제란 결론을 도출하기 위해 제공된 결론의 근거가 되는 명제이고 결론이란 전제에 의해 새로 유도된 명제를 말한다. 이렇듯 추론엔진의 추론과정은 전제와 결론간의 논리적 관계를 문제로 하여 해결하는데 그 역할을 둔다.

본 논문에서 제안하는 새로운 형태의 NPC 제어구조는 위에서 언급한 FSM과 추론엔진 방식을 결합한 하이브리드 방식으로 FSM만으로는 NPC의 지능적 행동제어가 어렵기 때문에 FSM과 추론엔진을 결합한 구조를 제안한다. 이러한 구조는 NPC의 행동은 크게 둘로 나누어 실시간의 제약을 받는 즉각 반응과 시간 제약은 덜한 대신 추론이나 계획 등을 수행해야 하는 지능으로 구성된다는 판단에 기초한다. 이것은 인간 역시 외부 자극에 빠르게 반응하는 반사 신경과 분석/추리/판단을 하는 지능, 이 두 가지 방식에 으로 행동을 결정하기 때문이다.

제안한 구조에서는 FSM과 추론엔진이 NPC 행동제어에 있어 그 특성별로 NPC 행동제어를 분담하도록 하였다. 이러한 역할 분담은 두 가지 관점에서 볼 수 있는데, 하나는 NPC 행동의 복잡성 측면이고, 다른 관점은 NPC 행동의 속성이라는 측면이다.

우선 NPC 행동의 복잡성 측면에서는, FSM은 간단히 유한개의 상태로 결정될 수 있는 행동양식을 구현하는데 사용하고, 추론엔진은 FSM으로 구현하기 어려운 행동양식을 구현하는데 사용한다. FSM은 계산 부담이 없고 유연성이 있기에 빠른 반응을 가능케 하고 간단한 반응의 추가가 쉽다. 따라서 자주 쓰이고 빠른 결정이 중요시되는 NPC의 행동제어에는 FSM을 이용하게 된다.

NPC 행동의 속성이라는 측면에서는, FSM은 행동양식을 결정하고, 추론엔진은 NPC의 추론능력을 나타낸다. 따라서 추론엔진은 주어진 상황으로부터 상태를 파악하는 보다 추상적인 역할을 하고, 상태가 파악된 이후에는 FSM을 사용하여 실제 NPC 행동을 제어하게 된다.

이러한 FSM과 추론엔진의 역할 분담은 다음과 같은 예에서 FSM만으로 구현하는 것에 비해 장점을 가질 수 있다. 위의 아이와 어머니 NPC의 예에서 FSM만으로 인공지능을 구현했다면 어머니 NPC는 사라진 아이 NPC를 찾기 위해 전체 공간을 순차적으로 탐색하기만 한다. 하지만 추론엔진이 사용된다면, 어머니 NPC는 이미 자신의 상황정보 속에 부근에 놀이터가 있다는 지식과 그리고 아이들은 놀이터를 좋아한다는 지식을 결합하여 사라진 아이 NPC가 놀이터에 있을 것이라고 예측하여, 다른 곳보다 우선 놀이터를 찾게 될 것이다.

또한, 제안한 구조에서는 FSM에 의해 결정된 행동이 추론엔진에 의해 번복될 수 있다. 왜냐하면 추론엔진에 의한 결정이 FSM에 의한 결정보다 지능적이므로 보다 합목적적 성격을 가지기 때문이다.

우리가 제안한 NPC 제어구조는 게임이나 가상환경의 오락성과 실감성을 증진시키기 위해 사용될 수 있다. 예를 들어, 축구 게임의 경우 시용자 플레이어는 한명의 캐릭터를 조작하게 되고 나머지 동료선수 NPC나 상대편 선수 NPC들은 축구의 규칙에 맞게 포메이션을 펼쳐가면서 동작하게 된다. NPC들의 단순히 공을 뺏으러 움직이는 행동은 FSM에 의해 제어되고, 이 외에 오프사이드 반칙을 유도하거나, 공격 시 동료 선수 NPC들이 지능적으로 작전에 따른 움직임을 보이게 하기 위해 추론엔진을 이용한다면 게임의 실감성을 향상시킬 수 있을 것이다.

## 2.1 제안하는 NPC 제어 구조

제안하는 NPC 제어구조는 그림 1과 같이 서버/클라이언트 구조에 기반을 둔다. 클라이언트에서는 그래픽적인 동작과 FSM을 수행하고 서버에서는 SWI-Prolog 기반 추론엔진 운영한다. 서버의 추론엔진은 클라이언트의 요구가 있을 때 클라이언트의 질의사항을 처리하여 답변을 다시 클라이언트에게 돌려준다.

FSM은 NPC가 보이는 즉각적인 반응을 제어하고, 계획 등의 복잡한 임무를 수행하게 하기 위해서는 추론엔진을 사용한다. 또한 게임이나 가상환경에서 진행 중에 발생하는 상태 정보들은 추론엔진에 fact 형태로 저장되고, 추론엔진은 이에 기반을 두어 해당 rule을 실행시켜 NPC들의 행동을 제어하게 된다. 이때 추론 결과의 산출물은 직전 FSM의 결과 산출물과 비교하여, 이미 FSM에 의해 제어된 행동과 동일하거나 비슷한 유형일 경우, 추론에 의한 결과는 무시된다. 하지만 결과 산출물이 다를 경우, 기존 행해지고 있는 FSM의 결과는 추론의 결과로 대체된다. 이렇게 함으로써, 추론엔진에 의한 결과물이 NPC의 행동을 최종적으로 제어하도록 하는 동시에, 추론 결과물이 나오기 전까지는 FSM의 결과를 사용함으로써 NPC 행동의 실시간성을 보장할 수 있게 된다.

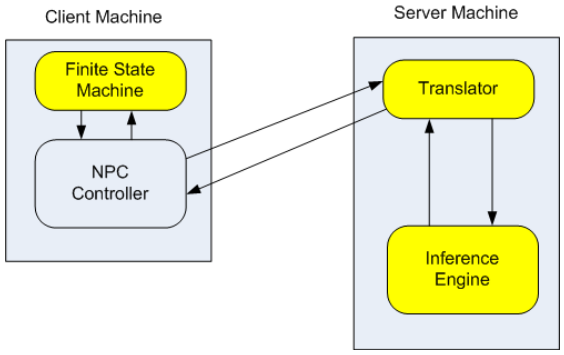


그림 2 제안하는 NPC 제어를 위한 FSM과 추론엔진의 하이브리드 구조

그림 1에서 translator의 역할은 크게 입력과 출력으로 구성된다. 입력 측면에서는 추론엔진으로 들어오는 데이터를 가공하여 효율적으로 저장할 수 있도록 하고, 출력 측면에서는 추론 결과물을 NPC 행동제어에 쉽게 이용할 수 있는 형태로 바꾸어주는 기능을 한다. 즉, 클라이언트는 FSM에 따라 동작하다가 필요에 따라 서버에 질의를 하게 되고, 이때 서버는 수신한 질의를 translator로 분석한 후 추론과정을 수행하고, 결과는 다시 translator를 통해 NPC 행동제어 명령어로 변환되어 클라이언트에게 전달된다.

또한, translator는 입력되는 정보들 가운데 중복된 데이터를 필터링하거나 aggregation하기도 해서, 데이터 처리 관련한 오버헤드를 감소시킨다.

서버/클라이언트 구조의 도입은 본 논문의 NPC 제어구조를 네트워크기반 시스템에 쉽게 적용 시킬 수 있도록 한다. 또한 NPC의 인공지능 부분이 수정 되어도 클라이언트 쪽의 수정 없이 서버의 인공지능 부분만 업데이트 해 주면 되기 때문에 보다 NPC 인공지능의 개발과정을 간단하게 하는 장점이 있다.

## 2.2 제안 구조의 구현

제안 구조의 추론엔진은 SWI-Prolog기반 추론엔진을 사용하여 구현하였다[4]. SWI-Prolog기반 추론엔진은 C 언어를 포함한 다른 프로그래밍 언어와의 연동을 위해 API를 제공하며, DLL파일로 제공되는 SWI-Prolog 라이브러리에는 SWI-Prolog접근을 위한 함수들이 있어 이들을 이용하여 추론엔진을 구현하였다. 예를 들어, 이러한 함수들을 이용해서 자체 구현한 QueryToProlog() 함수를 이용하여 추론엔진 내부의 모든 rule을 접근할 수 있다. 그림 2에서와 같이, QueryToProlog() 함수는 서버의 translator에서 사용되는데, 클라이언트에서 넘어온 질의 데이터를 추론엔진 입력형식에 맞도록 변화하거나, 추론 결과를 클라이언트에게 전송전 재 변환하는 역할을 한다.

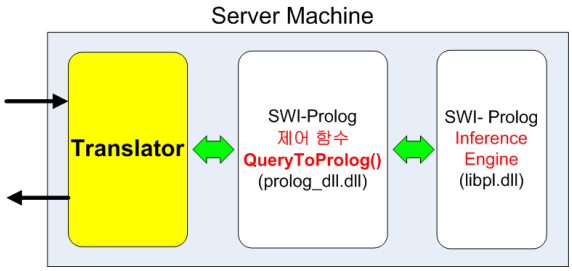


그림 3 QueryToProlog() 함수의 역할

서버는 다수 클라이언트의 접속을 처리해야 하고, 각 클라이언트에서 동시 다발적으로 발생하는 질의사항을 효과적으로 처리해야 한다. 이를 위해 서버의 translator에서는 연결된 각 클라이언트 요구사항 처리를 위한 개별 쓰레드를 생성한다.

Translator는 서버 추론 엔진에 대한 질의 제어 기능도 가지도록 구현된다. 원래 서버 추론엔진은 한 번에 하나의 질의에 대해서만 응답이 가능한 구조이다. 하지만 서버와 연결된 클라이언트가 다수이므로, 어느 한 순간에 하나의 클라이언트만 질의할 수 있도록 제어해야 한다. 이를 위해 그림 3과 같이 질의 큐 (Question Queue)를 사용하여 질의를 FIFO 형식으로 처리할 수 있도록 하였다.

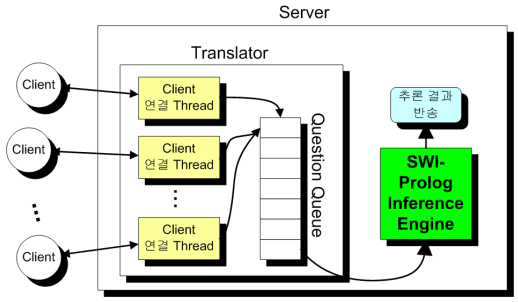


그림 4 Translator의 구조

서버와 클라이언트간의 네트워킹은 그림 4와 같이 Quanta library[2]를 사용하여 다양한 형태의 클라이언트와 서버간의 통신 QoS를 지원할 수 있도록 한다. 특히 Quanta-TCP, Quanta-Thread등을 사용해 클라이언트와 서버의 네트워킹을 구현하였다. 클라이언트는 다수의 NPC들이 수집한 정보들을 Quanta 네트워킹을 통해 서버에게 전송하고, 서버는 추론 결과를 클라이언트들에게 전송한다.

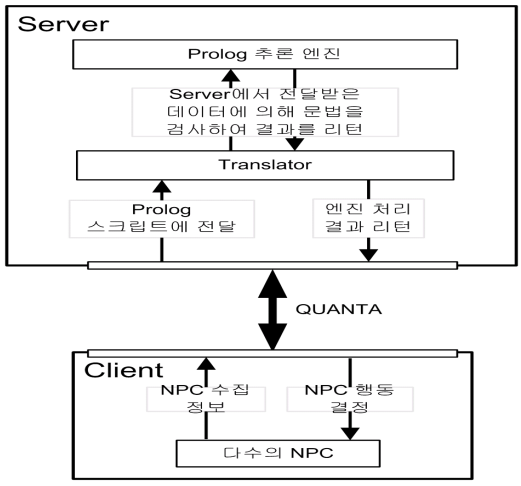


그림 5 Quanta를 이용한 서버와 클라이언트 네트워킹 연동

클라이언트와 서버간의 네트워크 메시지는 그림 5와 같은 메시지 구조로 전송된다. 패킷의 가장 앞 30바이트는 질의 대상이 되는 rule의 이름을 나타내고, 다음 4바이트는 총 인자의 개수, 그 다음 4바이트는 결과를 위해 필요한 인자의 개수를 나타낸다. 이 후 메시지는 rule에 필요한 인자의 데이터가 들어가게 되는데 인자는 스트링으로 구성되기 때문에 인자 하나당 30바이트씩 저장 공간을 가진다.

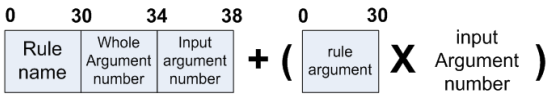


그림 6 네트워크 데이터의 패킷 구조

서버에서 추론엔진에 의해 나온 결과는 클라이언트에게 반송되어야 하는데, 이 때 서버에서 클라이언트로 보내는 메시지는 단순한 스트링 형태로 구성된다. 따라서 서버는 다양한 길이의 스트링 메시지를 여러 번에 걸쳐 전송할 수 있다.

## 2.3 데모를 위한 NPC 제어 방식 구현

데모를 위해 구현된 가상 전시공간의 가족관람 NPC들은 그림 6과 같이 FSM과 결과 추론을 위한 질의로 이루어진다. 본 데모에서는 아버지, 어머니, 아이의 NPC들이 사용되고, 각 NPC는 각자의 FSM을 가지고 장애물 피하기나 이동을 위한 경로 설정 등을 하게 된다. FSM 상태들 중에는 추론을 실행하는 상태가 있다. 이 상태에서는 SWI-Prolog 추론엔진에 질의를 하여 제어명령을 받아오게 된다. 추론엔진의 제어명령에 따라 FSM은 다음 동작을 행하게 된다. 각 NPC의 FSM은 서로 독자적으로 동작하지만 필요에 따라 추론엔진에 질의를 하며 협력하는 형태로 동작하게 된다.

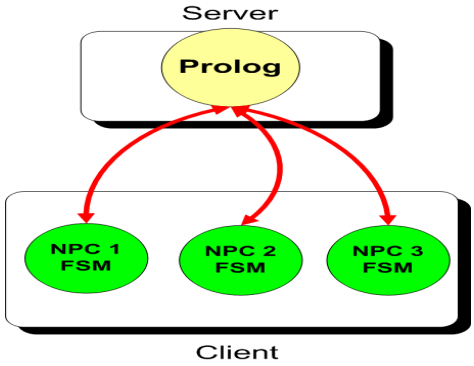


그림 7 SWI-Prolog를 사용한 NPC 제어

구현한 데모 시스템을 이해하기 위해서는 SWI-prolog 프로그래밍에 대한 이해가 필요하므로, 여기서 간단한 예를 들어 설명한다. 조부모(A, C)의 rule이 있고 조부모(A, C)는 부모(A, B), 부모(B, C). 로 구성된다. 그리고 부모(철수, 영희). 부모(영희, 길동)의 fact들이 있다. 이 경우 질의가 조부모(철수, X)라면 X에는 길동이 해당하게 된다. 즉 정의한 rule에 의해 fact들을 패턴 매칭한 결과로, 제공된 명제인 fact들을 사용하여 조부모() 라는 rule에 의해 X에 들어가는 명제인 '길동'을 유도한 것이다.

위의 예를 prolog 스크립트 언어로 기술하여 보면 아래와 같다.

```
# facts
parent(chul-soo, yung-hee).
parent(yung-hee, gil-dong).
# rule
```

```
grandparent(GP, GC) :-
  parent(GP, Parent),
  parent(Parent, GC).
```

grandparent(GP, GC) rule의 사용은 GP가 chul-soo라고 알고 있을 때 GC를 알기 위해 사용되거나 GC를 알고 있을 때 GP를 알아내는데 사용할 수 있다. 예를 들어 GP를 알고 있을 때 SWI-Prolog를 사용한 질의는 다음과 같다.

```
grandparent(chul-soo, GC).
```

GC에는 facts와 rule에 의한 적절한 답이 들어가게 되는데 facts에 의한 결과는 gil-dong이 된다.

데모 시스템에서의 추론엔진은 아버지, 어머니, 아이 각각의 행동을 제어하는 여러 개의 rule들을 수행하기 위해 사용된다. 아버지 NPC가 추론엔진으로 질의를 전송하는 경우는 아버지 NPC가 FSM의 어느 상태로 전이할지 결정할 수 없으나 이동할 준비가 되었을 때 발생한다. 이 경우 클라이언트는 서버에게 아버지 NPC가 관람한 전시물이 어떤 것인지 정보를 넘겨주게 되고, 추론엔진은 스크립트에 정의된 아버지 NPC의 전시물 선호도와 클라이언트에서 넘어온 전시물의 관람여부를 가지고 아버지 NPC가 다음으로 관람하게 될 전시물이 어떤 것인지 결정하여, 아버지 NPC의 관람행동을 제어한다.

어머니 NPC는 기본적으로 FSM에 의해 아버지 NPC를 따라 움직이게 된다. 이동 중에 아이 NPC가 어머니 NPC의 곁을 벗어나 다른 전시물을 보러 갔을 때 어머니 NPC의 FSM은 추론엔진에 다음 행동에 대한 질의를 하게 된다. 추론엔진은 어머니 NPC와 아이 NPC간의 거리에 대한 정보를 받게 되고 그에 따라 어머니 NPC의 목적지를 정해준다. 따라서 어머니 NPC는 아이 NPC를 보호하기 위해 아이 NPC 방향으로 이동하게 된다.

아이 NPC의 rule들은 기본적으로 아동들이 가지는 산만함을 표현한다. 아이 NPC는 어머니 NPC와 같이 FSM에 의해 아버지를 따라 이동하다가 주위 일정 범위 안에 관심 전시물이 있을 때, 행동에 대해 추론엔진에 질의를 한다. 추론엔진은 아이 NPC의 일정 범위 안에 있는 전시물에 대한 정보를 받아 아이 NPC가 해당 전시물 쪽으로 이동할 것인지를 결정한다. 만약 이동이 결정되면, 아이 NPC는 해당 전시물을 보기 위해 가족그룹에서 이탈하게 된다. 이러한 이탈행동은 어머니 NPC가 아이 NPC 보호를 위해 근접할 때까지 계속된다. 만약 어머니 NPC가 일정 시간이 지나도 아이 NPC를 보호하러 접근하지 않았고, 아이 NPC가 전시물을 충분한 시간동안 관람하였을 경우, 아이 NPC는 해당 전시물에 대한 흥미를 잃고 다시 가족그룹을 찾아 이동하게 된다.

위에 설명한 rule들을 SWI-Prolog로 구현한 것 들 중 특히 아이 NPC를 위한 SWI-Prolog 구현은 아래와 같다.

```
get_child_order(A, B, C, F_see, M_see, M_status,
Order):-
    M_status == call,
    Order = follow_mother;
    A == see,
    get_dest(a, Order);
    B == see,
    get_dest(b, Order);
    C == see,
    get_dest(c, Order);
    F_see == see,
    Order = go_around;
    M_see == see,
    Order = follow_mother;
    F_see == no,
    Order = follow_father.
```

아이 NPC의 행동을 모델링 한 위의 rule에서는 전시물 A, B, C의 관람 여부와, 아버지와 어머니 NPC들의 근접여부, 그리고 어머니 NPC의 상태에 따라 아이 NPC의 이동 순서를 결정하게 된다.

위 rule에 대한 클라이언트의 질의는 아이 NPC가 전시장을 가족과 함께 관람 중에 아이 NPC의 시야에 다른 흥미 있는 전시물이 들어왔을 때 발생한다. 클라이언트는 위의 rule인 get_chile_order()에 대한 질의를 하여 order의 내용에 따라 아이 NPC의 다음 동작을 결정한다.

Order의 내용은 '전시물 A, B, C 관람' 명령이거나 아버지 또는 어머니 NPC를 따라서 움직이게 하는 명령이다. 특히, 아이 NPC는 어머니 NPC의 상태가 아이 NPC를 찾는 상태가 아니고, 전시물 A, B, C중 관람하지 않은 전시물이 있다면 해당 전시물 방향으로 이동하게 된다.

아이 NPC가 해당 전시물로 이동하기 위해선 get_dest()를 호출하게 되는데, 해당 rule의 결과로 전시물 방향으로의 이동 명령을 받게 된다. get_dest()의 내용은 다음과 같다.

```
get_dest(Exh, Dest):-
  Exh == a,
  order(a, Dest);
  Exh == b,
  order(b, Dest);
  Exh == c,
  order(c, Dest).
```

## 3. 제안 구조를 이용한 데모 시연

본 논문에서 제안한 NPC 제어구조의 데모 시연을 위해 앞장에서 설명한 바와 같이 가상 전시공간을 관람하는 3인 가족 NPC의 행동양태를 구현하였다. 서버/클라이언트 구조로 이루어진 데모는 하나의 서버와 다수의 클라이언트로 구성되는데, 클라이언트에서는 그래픽 환경, NPC의 FSM 제어와 네트워킹을 통한 서버로의 질의를 한다.

본 데모의 NPC 제어 중, 장애물 피하기나 경로 설정 등은 FSM 기반으로 구현하였으며, 어린이 NPC의 산만한 행동양식이나 어머니 NPC의 행동양식은 SWI-Prolog 엔진기반 추론엔진으로 구현하였다. 그래픽 환경은 Apocalyx 엔진[5]기반으로 구현하였다.

3인 NPC 가족의 가상 전시 공간 관람 제어는 어린이 NPC와 부모 NPC로 이루어진다. 3인 가족 중 어린이를 나타내는 NPC와 부모를 나타내는 NPC는 가상 전시공간에 존재하는 3개의 전시물을 관람하게 된다. 어린이 NPC는 아동들이 흔히 가지는 산만함을 가지고 있어, 부모 NPC들은 아이 NPC가 홀로 떨어지지 않도록 동작한다.

데모에서는 기본적으로 아버지 NPC를 기준으로 가족이 그룹을 지어 움직이게 되는데, 아버지 NPC는 여러 개의 전시물 가운데 자신이 흥미 있어 하는 전시물을 먼저 관람하게 된다. 도중 어린이 NPC는 근처에 있는 전시물을 홀로 관람하러 움직이게 되고, 이 때 어머니 NPC는 아이 NPC가 일정 범위 밖으로 없어진 것을 알게 되면 어린이 NPC를 찾는 행동을 시작한다.

구현된 데모의 동작은 다음과 같다. 그림 7은 크기와 색깔별로 구분된 가족 NPC의 초기 모습을 보여준다. 가장 큰 NPC는 아버지, 그 다음은 크기순으로 어머니와 아이 NPC를 나타낸다. 그림 8은 관람도중 아이 NPC가 혼자 이탈하는 모습을 보여주며, 그림 9는 이탈한 아이 NPC를 다시 데리고 가는 어머니 NPC를 보여주고, 그림 10은 다시 가족그룹을 형성하여 관람을 계속하는 모습을 보여준다.

## 4. 결론

본 논문에서는 FSM과 추론엔진을 결합한 NPC 행동제어 구조를 제안하고, 제안된 구조에 기초한 데모를 보였다. 이러한 제안 구조는 FSM이 제공하는 실시간성을 유지하는 동시에 추론엔진이 제공하는 보다 지능적인 NPC 행동양식을 만들어 낼 수 있다는 장점이 있다.

## 참고문헌


[1]http://www.aistudy.com/

[2]http://www.evl.uic.edu/cavern/quanta/

[3]Mat Buckland, "Programming Game AI by Example", Wordware Publications

[4]http://www.swi-prolog.org

[5]http://apocalyx.sourceforge.net


그림 8 데모 – 가족 그룹의 관람 초기상태

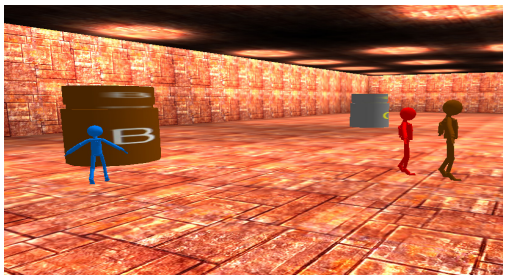

그림 9 데모 – 어린이 NPC의 이탈

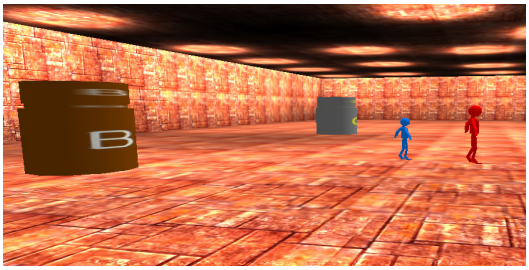

그림 10 데모 – 이탈한 아이 NPC를 찾는 어머니

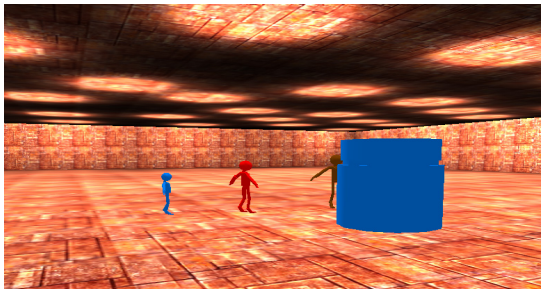

그림 11 데모 – 가족 그룹의 재형성