제26회 한국정보처리학회 추계학술발표대회 논문집 제13권 제2호 (2006. 11)

정형 기법을 적용한 해양 NPC 모델 설계 및 구현

김종한* 정승문* 김병기**
*동신대학교 디지털콘텐츠협동연구센터, **전남대학교 전산학과
email : chkim@dsu.ac.kr

Design and Implementation of Oceanic NPC Model applying Formal Method

Kim Chong Han*, Jeong Seung Mun*, Kim Byung Ki**
*Dongshin University Digital Contents Cooperation Research Center
**Chonnam National University Dept. Computer Science

요 약

NPC(Non playable Character)모델은 온라인 게임뿐만 아니라 가상공간 시스템 구축 시 빠질 수 없는 중요한 요소이다. 현재 가장 널리 사용되는 인공지능 처리방식의 하나인 FSM(Finite State Machine)은 NPC의 행동양식을 표현하기 위해 유한한 개수의 상태를 이용하는 알고리즘이다. 인공지능이 적용된 NPC 모델 설계시 정확한 명세는 구현 단계에서 발생되는 자원의 손실을 막아주고 요구명세에 따른 검증을 가능하게 한다.

본 논문에서는 해저가상공간 구축 시 발생되는 어류 객체의 행동패턴을 분석하여 속성을 정의하였으며, 환경변화에 따른 행동 특성의 상호관계를 설정하여 정형화하였다. 정의된 속성을 가진 NPC 모델을 FSM 알고리즘을 적용해 설계하고 구현한다. 설계된 NPC모델은 CTL기반의 모델체커인 SMV(Symbolic Model Verification)를 통해 검증함으로써 설계에 대한 타당성을 입증하였다.

1. 서론

해저가상환경을 구축하는데 있어서 어류 등의 동적객체들에 부여하는 AI기능은 해저생태환경을 구현하는 중요한 부분을 차지한다. 다양한 종류의 해저생물들에 각각의 특성 및 기능을 정의하고 이에 따라 객체를 설계함으로써 해저에서 발생되는 먹이사슬, 생식, 행동유형 등과 같은 상황을 사실적으로 표현할 수 있다. NPC모델을 설계하기 위한 인공지능 알고리즘으로는 유한 상태 기계를 이용한 FSM알고리즘, 학습을 통한 진화을 이용한 학습알고리즘, 유전자 및 신경망 알고리즘, 집단행동을 표현하기 위한 Flocking 기법, A* 알고리즘을 통한 길 찾기 알고리즘 등을 중심으로 연구되어 왔다.[1] FSM알고리즘은 가장 널리 사용되는 인공지능 처리방식이며 장치나 어떤 기술의 순차 로직 또는 제어기능에 단순하면서도 정확한 설계를 가능하게 한다. 그러나 상태의 수에 제한이 없기 때문에 복잡하고 많은 수의 상태표현을 요구하는 경우 상태 다이어그램을 정리하기 어렵고 상태 변화를 위한 외부 자극 처리가 복잡해지게 된다. 이를 위해 유한 상태 시스템에서 그 모델이 만족해야할 특성을 시제논리로 명세하여 오류의 존재 여부를 판단하는 모델체킹의 도구인 SMV를 사용하여 모델의 정확성을 검증한다.

본 논문에서는 해저 어류에 대한 행동패턴 및 특징을 정의하고 FSM 알고리즘을 통해 NPC 모델을 설계하고 구현한다. SMV를 통해 설계된 알고리즘의 정확성을 검증한다. 2장에서는 NPC 모델에 대해 알아보고 모델체킹 및 CTL(computer tree logic)에 대한 설명한다. 3장에서는 어류의 행동 패턴 및 생태, 특징의 정형화하여 정의하고 이를 FSM알고리즘으로 설계한다. 또한 설계된 알고리즘을 SMV 언어로 변화하여 설계에 대한 정확성을 검증한다. 마지막으로 4장에서는 결론 및 향후 연구방향에 대해 설명한다.

2. 관련연구

현재 온라인 게임을 중심으로 가상공간 및 PC(playable Character), NPC 모델에 대한 연구가 활발히 이루어지고 있으며 또한 더욱 지능화된 NPC모델의 구현을 위한 노력이 계속되고 있다. 사용자가 직접 제어하는 PC와 달리 NPC 모델은 컴퓨터가 직접 제어하는 모델이다. 이것은 가상 공간 내에서 주어진 이벤트에 반응하거나 어떠한 조건에 대해서 특정한 상황을 연출한다. 다양한

상황을 연출하기 위해 비교적 구조가 간단한 FSM알고리즘을 이용하여 NPC모델을 설계하는데 조건이 복잡해지고 설정된 상황이 많으면 모델 설계에 어려움이 많을 뿐만 아니라 설계된 모델을 수정 및 개선하는데도 많은 시간과 노력이 필요하다. 또한 잘못된 설계로 인한 자원의 손실을 감수해야 된다. 이는 설계에 대한 검증의 필요성을 제기한다.

2.1 NPC(Non-Playable Character)

NPC 모델이란 사용자의 어떠한 제어에도 종속적이지 않으며 스스로 판단하여 동작하는 가상공간내의 동적인 객체를 뜻한다. NPC는 기본적으로 개발자가 정의한 제어구조와 상태천이 정보를 바탕으로 수행된다. 이는 개발자가 설계 시 고려해야 될 중요한 요소이다.

일반적으로 NPC는 인지(sense), 사고(think), 행동(act), 결정주기(decision cycle)을 반복하다.[3]

표1은 각 단계에서의 활동을 보여준다.

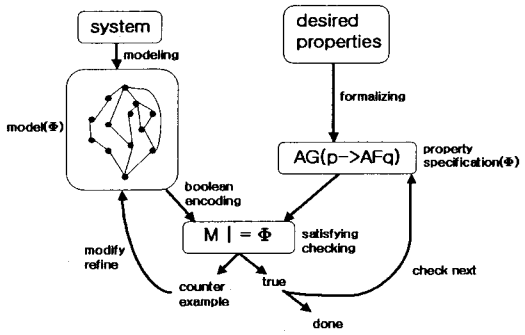
인지 단계에서는 서버로부터 현재 위치에서 발생되는 정보를 감지하여 상황을 파악하고 사고 단계에서는 현재 상황에서 자신의 역할에 부합되는 행동을 결정하고 마지막으로 행동 단계에서는 택된 행동을 수행하도록 서버에 명령한다. 각 단계는 순환한다.

2.2 Model Checking

시스템의 검증 방법은 비정형적인 방법과 정형적인 방법으로 나눌 수 있다. Simulation과 Test등은 비정형적인 방법이며, 정형적인 방법으로는 Theorem Proving, 모델 체킹 등이 있다. 코드 기반 모델체킹은 추가적으로 시스템의 정형 명세가 불필요할 뿐만 아니라 구현된 결과물에 대하여 검증을 수행하기 때문에 구현과정에서의 오류를 더 검출할 수 있다. 이로 인하여 소프트웨어의 신뢰도를 더욱더 향상 시킬 수 있을 뿐 아니라 정형 기법의 비전문가인 개발자들에 의해서도 쉽게 적용될 수 있는 장점을 가진다.

[표 1] 시뮬레이션과 모델 체킹의 비교

	종류	장단점
정형검증	Simulation	• 규모가 큰 설계에서 다루기가 힘들다. • 희박한 에러까지 검출이 가능하다. • 모든 상태를 확인한다.(상태폭발을 야기)
비정형검증	Model checking	• 규모가 큰 설계에서 다룰 수 있다. • 발견하기 힘든 곳의 에러(corner-case bugs)를 찾아내기가 어렵다. • 모든 상태를 확인하기는 불가능하다.



(그림1) Model Checking Flow

- Input
  - M : finite-state model
  - Φ : logical formula

Input 값은 시스템의 유한상태모델(M)과 요구사항을 형식화한 논리형식(Φ)이다.

- Decision
  - M | = Φ (M satisfies Φ)

Input값이 확정되면 시스템이 요구사항을 만족하는지를 검사한다.

- Output
  - true
  - false (with counter examples)

Output은 만족하면 true, 만족하지 목하면 false를 출력한다.

2.3 CTL

CTL[1]은 시간의 흐름을 여러 갈래의 트리 형태로 표현한다. 즉, 특성을 계산 트리의 형태로 표현한다. 즉, 초기 상태로부터 무한한 경우를 나타내는 트리 형태로 표현하는 상태 그래프를 이용하여 시스템의 동작을 명세하는 방법이다. 이에 비해 LTL은 시간의 흐름을 시간을 선형적으로 표현하는 방법이다. 따라서 CTL과는 달리 시스템의 동작이 하나의 시퀀스로 표현된다.

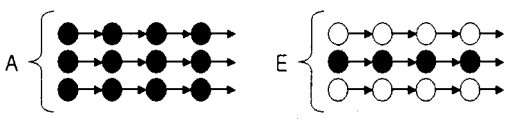
CTL은 경로 한정자(Path Quantifier)와 시제 연산자(Temporal Operator)의 조합으로 이루어진다.

- CTL(Computer Tree Logic)

  Path Quantifier(A, E) +Temporal Operator(X, F, G, U)

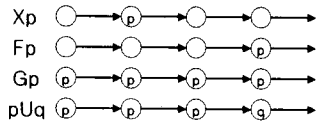
A : 모든 경로대해서 (For all paths)

E : 최소한 하나의 경로에 대해서 (There exists a path)



(그림 2) 경로 한정자(Path Quantifier)

A는 모든 경로에 대해서 다 만족해야 참이다.
E는 최소한 하나의 경로에 대해서만 만족하면 참이다.
Xp : 다음시점에 p가 나와야 참이다. (Next)
Fp : 언젠가 p가 나오면 참이다. (Future)
Gp : 언제나 p가 나와야 참이다. (Globally)
pUq: q가 나오는 시점까지 계속 p가 나와야 참이다.(Until)



(그림 3) 시제 연산자(Temporal Operator)

3. 해양객체의 FSM 설계 및 NPC 구현

3.1해양객체 행동패턴 정의

저 수중객체는 다양한 종들이 존재하지만 본 논문에서는 어류에 대한 특징만 규정짓도록 한다. 어류의 속성 중 중요한 몇 가지를 구분해보면 먹이사슬, 성장상태, 서식환경, 성별, 등을 들 수 있고, 이 속성에 따른 행동으로 생식, 회피 등을 들 수 있다. 표2는 각 객체들이 가져야할 속성 및 행동을 설명한다. 본 논문에서는 아래 표에 나타나있는 6가지 어류에 대해서만 언급하도록 한다.

<표2>각 객체들의 속성 정의

종(A)	Grade	A1 : 넙치, A2 : 성대 A3 : 상어, A4 : 꽁치 A5 : 도미, A6 : 가오리
	note	각 객체에 대해 종명 정의
먹이사슬 관계 (B)	Grade	B1 ~ B10 등급
	note	각 객체는 자신 보다 3등급 아래의 객체를 포식할 수 있음 B1은 최하위 피식객체 ~ B10은 최상위 포식객체임 같은 종의 경우 포식관계가 성립하지 않음 성장상태가 먹이의 5 등급 이상일 경우 포식이 가능함.
성장 상태(C)	Grade	C1 ~ C20
	note	각 등급은 6개월 차이
서식 환경(D)	Grade	D1 ~ D5
	note	바닥층을 D1, 해수표현을 D5로 주어 5단계로 구분
성별(E)	note	F1 : Male F2 : Female
번식(F)	note	생식은 같은 종에 대해서만 가능. 생식은 항목이 C5이상 가능
행동(G)	Grade	G1 ~ G5
	note	G1은 느림, G5 빠름으로 5단계로 구분

속성표에서 A~E까지는 객체들이 가져야 할 속성부분을 정의한 것이고 F와 G는 활동을 정의한 것이다. 각 객체에 대해 속성 및 행동을 입력한다. 구축하고자 하는 해저 가상해저의 특성상 포식자에 대한 조건은 배제하고 적에 대한 회피 조건만 고려한다. 즉 상어는 더 낮은 계층의 객체에 대해 포식에 관한 어떠한 행동을 하지 않고, 낮은 등급의 객체는 상어와 같은 높은 계층의 객체에 대해 회피와 같은 행동을 한다.
해저 가상 공간 내 삽입될 일부 객체에 대한 행동패턴은 다음과 같이 정의한다.

<표 3> 객체들의 속성부여

Species	Property	Activity
Object1	A1, B3, C5, D1, E1,G2	F, G1~G3
Object2	A1, B3, C2, D1, E2.G2	F, G1~G3
Object3	A3, B9, C10, D2~D4, E1, G2	F, G1~G4
Object4	A5, B3, C6, D2, D3, E1, G2	F, G1~G4
Object5	A4, B2, C2, D3, D4, E2, G2	F, G2~G5
Object6	A6, B6, C9, D1, D2, E2, G2	F, G1~G3
Object7	A1, B3, C8, D1, E2, G2	F, G1~G3

3.2 FSM을 이용한 해저 NPC모델러 설계 및 구현

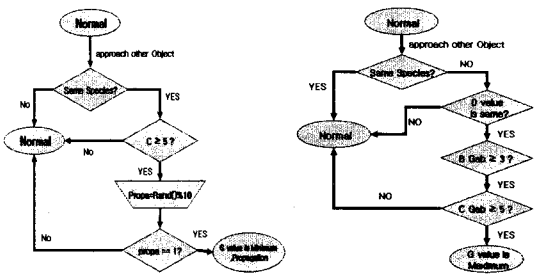
표4는 각 행동을 미리 정해진 속성에 맞게 정형화시킨 간단한 예이다. 번식의 경우 같은 종이고 성장상태가 5이상이면 10%확률로 번식하면서 움직임이 둔해진다. 행동의 경우 다른 종이 행동반경에 인지되었을 때 서식공간(D)이 같고, 먹이사슬단계(B)의 차이가 3이상이고 성장상태(C)의 차이가 5이상이면 G값은 최대값을 갖는다.

<표 4> 행동패턴 Pseudocode

생식	활동

If(Same Species == true {
else if(C≥5))
{Propa = Rand()%10
if(propa = 1)
F="yes";
"G = Minimum Value";}}
else {no state transfer}
	If(Same Species == true){{{
	else if(Same D ==true)}
	else if(Gap B ≥ 3)}
	else if(Gap C ≥ 5),
	G = Maximum Value}
	else{no state transfer}

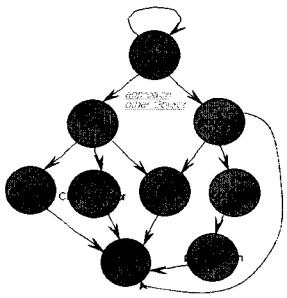
그림4는 위에서 제시된 조건에 대한 행동을 표현한 FSM 플로우차트이다. 번식의 경우 Normal 상태는 어떠한 조건도 입력되지 않는 안정된 상태를 말한다.



(그림 4) 번식 및 먹이사슬 플로우차트

3.3 설계된 FSM의 검증

FSM의 검증은 SMV를 통하여 이루어진다. 어떤 조건이 입력된다면, 한 상태는 다른 상태로 천이. 잘못된 초기 설계는 발생해서는 안 되는 상태를 발생시킬 수 있다. 이는 시스템의 심각한 오류를 불러일으킨다. 그림5은 Active HDL이용한 하나의 특정 객체에 대한 상태들의 초기 설계이다.



(그림 5) 객체의 FSM

표5는 FSM을 SMV 코드로 변환한 것이다.

<표 5>객체의 상태를 표현한 SMV코드

```
MODULE main
 VAR
        state : {Stable,Fear,Goodwill,Evasion,Changecolor,
Hide,Courtship,Propagation,Stable1};
 ASSIGN
        init(state) := Stable;
        next(state) := case
        state = Stable : {Fear, Goodwill, Stable};
        .... skip ....
        state = Courtship : Propagation;
        state = Propagation : stable1;
        1 : state;
 esac;
```

몇 가지 설정을 통해 초기설계에 대한 오류가 없는 지를 확인할 수 있다.

조건	예상 결과치
① 객체가 두려움(Fear)를 느끼면 숨을 수도 있다.	True
② 객체가 구애(courtship)를 하면 다음 상태에 반드시 번식을 해야한다.	True
③ 객체가 호감(goodwill)을 가지면 숨는다.	False
④ 객체가 호감을 가져도 아무런 반응을 하지 않을 수도 있다.	True

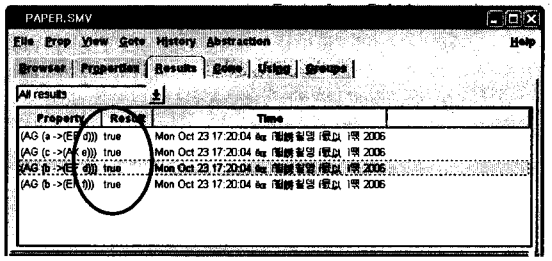
표6은 위 조건을 SMV코드로 변환한 것이다.

<표 6> 설정에 따른 SMV 코드

```
DEFINE
a := state = Fear;
b := state = Goodwill;
c := state = Courtship;
d := state = Hide;
e := state = Propagation;
f := state = Stable1;

SPEC AG(a -> (EF d))
SPEC AG(c -> (AX e))
SPEC AG(b -> (EF d))
SPEC AG(b -> (EF f))
```

그림6는 설정을 검증한 결과이다. 그림안의 둥근 원안의 내용은 결과 값를 나타낸다. 3번 조건의 경우 예상결과치가 false가 나와야 하는데 검증결과 true 값이 나왔다. 이는 설계에 오류가 있었음을 나타낸다. 즉 잘못된 설계이므로 검토가 필요하다.



<그림 6> SMV를 통한 검증결과

4. 결론

논문에서는 NPC 모델 표현을 위한 객체에 대한 속성 값을 정의였고 이를 FSM 알고리즘으로 설계하였다. 가상해저 공간 내에 NPC모델들을 삽입하였다. 삽입된 객체는 가상 공간 내에서 속성 값에 맞는 행동을 하는 것을 알 수 있었다. 또한 FSM알고리즘은 SMV를 통하여 검증하여 초기 설계에 대한 오류를 발견할 수 있다. 이것은 복잡해져가는 설계에 대한 개발자들의 부담을 줄일 수 있을 것이다. 향후 더욱 지능적인 NPC 모델의 설계 및 구현을 위해 속성의 세분화와 정형화가 필요할 것이다.

5. 참고문헌

[1]http://gamecode.org/article.php3?no=1588&page=0&current=ai&field=tip

[2]방기석, 이주용, 최진영, "SPIN을 이용한 CTL 모델체킹 방법론 연구," 한국 정보처리학회 춘계 학술발표논문집 Vol.08 No.01, 2001.

[3]http://en.wikipedia.org/wiki/Non-player_character

[4]권기현 "모델체킹의 이해", 소프트웨어 교육 세미나자료집, May 2003.

[5] "Model Checking Large Software Specification", IEEE Transactions on Software Engineering, Vol.24, No.7, July 1998.

[6]E. M. Clarke, O. Grumberg, and D. E. Long, "Checking and Abstraction", In Proceedings of the Nineteenth Annual ACM Symposium on Principles of Programming Languages, January, 1992.

[7]Gihwon Kwon, TaeHo Eom, "Equivalence Checking of Finite State Machines with SMV", Korea Information Science Society, Vol.30, No7, pp.642~648, 2003.8.