Statechart 상호 변환을 위한 Semantics의 분석

박승현*O 황대연* 이나영** 김윤구*** 최진영*
고려대학교 정보통신대학 컴퓨터학과
{shpark^O, dyhwang, choi}@formal.korea.ac.kr*

서울대학교 공과대학 원자핵공학과**
grasia2@snu.ac.kr**

삼창 엔터프라이즈***
ygkim@secrnd.co.kr***

An Analysis of Semantics for Transformation of Statechart

SeungHyun Park*O, Dae Yon Hwang*, Jin-Young Choi*
Dept. of Computer Science and Engineering, Korea University*

Na Young Lee**
Dept. of Nuclear Engineering, Seoul National University**

Yun Goo Kim***
Samchang Enterprise***

요 약

초기에 제안된 Harel의 statechart를 개선하기 위한 노력으로 다양한 statechart의 변형들이 생겨나고, 시스템 개발자는 대상 시스템에 맞는 적절한 설계 명세 언어와 자동화 도구를 선택하는 것이 중요하게 되었다. 대상 시스템에 맞는 명세 언어를 선택하고, 다양한 자동화 도구(CASE tool)의 기능을 적용하기 위해 statechart의 상호 변환을 통한 각 지원 도구의 적용이 필요하다. 하지만, statechart 상호 변환 과정에서 각 statechart의 의미론적(semantics) 차이는 의도하지 않은 오류를 야기할 수 있다. 따라서 본 논문에서는 이들의 의미론을 비교 분석하기 위한 몇 가지 간단한 예제를 시뮬레이션 해 보고, statechart간 상호 변환 방안의 기틀을 마련한다.

1. 서 론

Statechart는 상태 기반(state-based)으로 시스템의 행위를 모델링하기 위한 직관적인 명세 언어이다 [1]. 이러한 statechart는 Harel에 의해 제안되어 함수 지향(function-oriented)의 구조적 분석 패러다임(structured-analysis paradigm)을 갖고 처음 개발되었다 [2, 3]. 이후, 소프트웨어 기술이 발전하고 개발 패러다임이 변화하면서 객체 지향(Object-Oriented) 개발 방법론이 등장하였고, statechart의 의미론도 패러다임의 변화와 함께 발전을 거듭하며 다양한 statechart 변형들이 생겨나게 되었다. 따라서 소프트웨어 개발자들이 개발해야 하는 소프트웨어의 성격에 맞는 적절한 프로그래밍 언어를 선택해야 하는 것처럼, 시스템 개발자 역시 대상 시스템에 맞는 적절한 설계 명세 언어를 선택하는 것이 중요하게 되었다. 또한 statechart와 이를 지원하는 자동화 도구들의 특성, 적용 범위가 각각 다르기 때문에, 각 지원 도구에 따른 기능상의 보완을 위해 각 statechart간의 상호 변환이 필요하다. 하지만 각 statechart에 따른 각기 다른 특성과 구문/의미론적 차이가 존재하기 때문에 statechart간 변환에도 어려움이 있고, 변환 중에 예기치 못한 결과가 발생할 수 있다. 본 논문에서는 이들의 특성을 살펴보기 위해 의미론적 차이를 분석하기 위한 몇 가지 간단한 예제를 시뮬레이션 해 보고, 각 statechart간 의미론의 차이를 비교 분석을 통해 상호 변환 방안의 기틀을 마련한다.

논문의 구성은 2장에서 statechart의 일반적인 특성과 각 statechart에 따른 특성을 제시하고, 3장에서 각 예제에 대한 시뮬레이션 결과를 통해 각 statechart의 의미론적인 특성을 비교 분석하며 4장에서 결론과 향후 연구 과제를 제시한다.

2. 관련연구 : 각 statechart에 대한 overview

2.1 Statechart

Harel에 의해 처음 제안되었던 statechart는 실시간 시스템의 행위를 명세하는 비주얼 언어이다 [4]. 이러한 statechart의 특성을 정리하면 아래와 같다 [5].

statechart = state transition diagram + depth(hierarchy) + orthogoanlity(concurrency) + broadcast-communication

statechart는 유한 오토마타와 같이 상태와 이들 상태를 변화시키는 전이로 구성된 상태 전이 다이어그램(state transition diagram)에 기반한다. 여기에 계층성(hierarchy)을 도입하여 각 상태들은 하위 상태와 내부 전이(internal transition)을 가질 수 있고, 이들 상태는 각각 동시성(concurrency)과 독립성(orthogonality)을 지원하기 위한 AND-state와 순차적이며 배타성(exclusiveness)을 지원하는 OR-state의 형태를 갖고 있다 [6]. 또한 원 상태(source state)와 목적 상태(target state) 간의 관계를 표시하는 전이는 trigger [guard] / action으로 표현되는데, 이는 모든 statechart에 걸쳐 브로드캐스팅 되는 내부 event에 의해 통신할 수 있다. 트리거 이벤트는 전이를 발생시키고, 이에 따른 action으로 또다른 내부 event가 발생된다.

2.2 Statechart 의미론

처음 Harel에 의해 statechart가 제안된 이후, 소프트웨어 개발 패러다임의 변화와 함께 statechart의 구문과 의미론를 개선하기 위한 많은 노력과 연구 결과에 의해 statechart는 변화를 겪어왔다. 이에 따라 statechart는 여러 가지 변형이 생겨나고,

이들을 지원하는 다양한 자동화 도구에 의해 대상 시스템이나 적용 범위, 의미론에 따른 각기 다른 특성을 갖게 되었다.

우선 초기에 Harel에 의해 제안되어 현존하는 수많은 statechart 변형들의 기반이 되고 있는 오리지널 버전의 statechart를 classical statechart라 한다. 구조적 분석을 기반으로 정의된 classical 버전의 statechart는 아래와 같은 특성을 갖고, 이는 i-Logix의 STATEMATE에 의해 구현되었다 [7].

(1) 내부/외부 이벤트에 의한 반응(static reaction)과 스텝에 의한 변화는 스텝이 완료된 후에 발생한다.
(2) 이벤트는 오직 한 스텝에만 살아있다
(3) 스텝을 평가하는 guard는 스텝의 초기 상태에 기반한다.

또한 위의 classical statechart를 객체 지향 기반으로 정의한 객체지향 버전의 statechart는 아래와 같은 특성을 갖고, 이는 i-Logix의 Rhapsody에 의해 구현되었다 [8].

(1) 여러 이벤트가 한 스텝에 동시에 발생해도 해당 스텝이 완료될 때까지 이벤트 큐에 남아서 순차적으로 처리된다.
(2) 버퍼링을 사용하지 않기 때문에 action에 대한 결과가 부작용(side effect)을 발생시킬 수 있다.

본 논문에서는 각 statechart를 구분하기 위해 STATEMATE에 의해 구현된 Harel의 초기 statechart 버전을 Statemate statechart, Rhapsody에 의해 구현된 객체지향 버전의 statechart를 Rhapsody statechart, UML 정의 문서에 표현된 state machine을 UML statechart이라고 각각 표현한다.

3. 의미론적 차이

3.1 Choice

여러 복합 전이로 구성된 어떤 스텝의 내부에서 트리거에 따른 action의 결과가 해당 전이의 guard에 영향을 미치는 경우, 이를 dynamic choice라고 하고, 최초의 평가에 따라 그대로 전이가 일어나는 경우를 static choice라고 한다.

UML statechart에서 복합 전이는 junction vertex와 choice vertex를 이용하여 구현할 수 있다. 특히 단일 전이를 서로 다른 guard 조건으로 나누는 분기점으로 사용할 경우, junction vertex는 static conditional branch가 된다. 한편, choice vertex는 트리거의 guard를 동적으로 평가하여 전이를 일으키므로 dynamic conditional branch가 된다 [9].

1) Static Choice

그림 1은 Rhapsody와 STATEMATE의 static choice를 실험하기 위해 junction vertex를 이용한 예제이다. x의 값이 0인 State A1에서 event에 의한 발생하는 action이 전이 과정에서 수행되는지, 완료 후에 수행되는지의 여부에 따라 A2 또는 A3로 전이될 수 있다 [2].

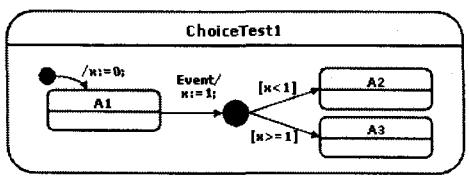

그림 1. Choice Test 1 (UML Statechart)

Rhapsody와 STATEMATE의 statechart에서 condition connector는 UML state machine에서 junction vertex와 같이 모두 static choice의 의미를 반영한다. Event에 따른 action이 트리거의 guard에 영향을 미칠 수 있지만, 트리거의 조건은 실제 스텝이 발생하기 전에 평가되고 action의 수행은 전이가 일어난 후에 이루어지므로 condition connector를 이용한 두 statechart 모두 static choice와 같이 전이가 일어난다.

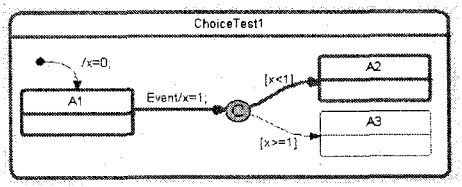

그림 2. Condition connector를 이용한 Choice Test I

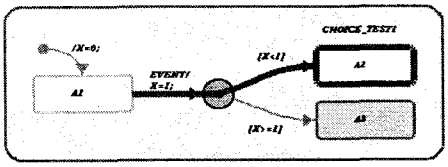

그림 3. Condition connector를 이용한 Choice Test I

그림 2와 그림 3은 각각 condition connector를 이용하여 static choice를 명세하고 시뮬레이션 한 결과이다. 초기 상태로 x의 값이 0인 상태 A1에서 Event가 발생하면, x의 값을 확인하여 guard [x<1]에 의해 전이가 발생한다. 그림 2, 그림 3 모두 상태 A2로 전이가 일어나고, 스텝의 마지막 단계에서 action을 수행하여 x에 1을 배정한다. static choice의 수행 과정을 그림으로 나타내면 아래와 같다 [7].

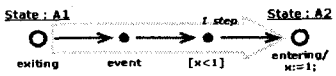

그림 4. static choice 수행 과정 (UML, Rhapsody, STATEMATE)

2) Dynamic Choice

UML statechart가 그림 5과 같이 명세되어 있다면, 이는 그림 1과는 달리 dynamic choice를 의미한다. x의 값이 0인 상태 A1에서 event가 발생하면, guard를 평가하기에 앞서 action을 수행하여 x에 1을 배정한다. 변화된 x의 값에 의해 guard를 평가하기 때문에 결과는 A3으로 전이하게 된다. 이와 같이 한 스텝을 수행하는 과정에서 action에 의해 트리거가 영향을 받아 동적으로 변화된 값에 의해 전이를 결정하게 되면, 이를 dynamic choice라고 한다.

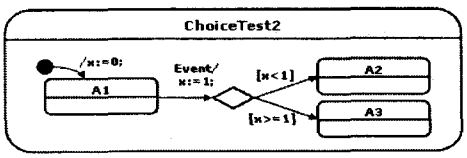

그림 5. Choice Test 2 (UML Statechart)

Rhapsody와 STATEMATE statechart는 앞서 설명한 바와 같이 기본적으로 static choice의 의미론을 따른다. 하지만 Rhapsody는 microstep을 이용하여 dynamic choice를 구현할 수 있다. 다음은 Rhapsody의 microstep을 이용하여 dynamic choice를 실험하기 위한 예제이다 [10].

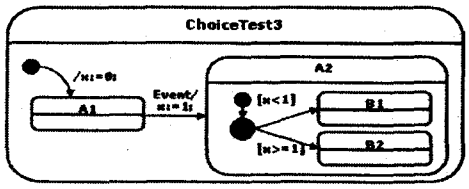

그림 6. Choice Test 3 (UML Statechart)

그림 6을 Rhapsody에서 명세하여 시뮬레이션 한 결과는 그림 7과 같다. 이는 microstep을 이용하여 dynamic choice를 구현한 것이다.

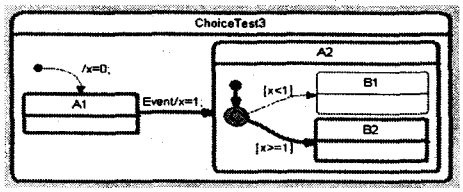

그림 7. Choice Test3 - Microstep (Rhapsody Statechart)

상태 A1에서 x의 값은 0이었지만, event가 발생하여 A2로 전이가 일어나고, A2의 초기 전이 과정에서 x의 값은 1로 갱신된다. 상태 A2의 condition connector에서 guard는[x>=1]이기 때문에 상태 B2로 transition 된다. 그림 7의 Rhapsody statechart, ChoiceTest3가 상태 A1에서 B2로 전이되는 과정은 아래의 그림 8, 그림 9와 같다.

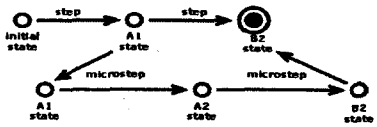

그림 8. Choice Test 3의 microstep 세부 과정 (Rhaposdy) [8]

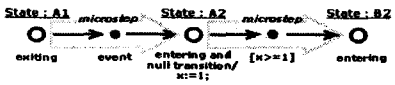

그림 9. Choice Test 3의 microstep 세부 흐름 (Rhaposdy)

한편, STATEMATE statechart의 스텝에는 microstep이 없기 때문에 하위상태를 사용해도 모두 static choice로 동작한다 [11]. 전이가 일어나기 전에 해당 스텝에서 고려해야 하는 모든 트리거를 평가한 후, 그에 따라 전이하게 된다. 따라서 그림 10은 그림 3과 같이 상태 A1에서 EVENT와 guard [x<1]에 의해 B1로 전이된 후, x에 1을 배정하며, 과정은 그림 11과 같다.

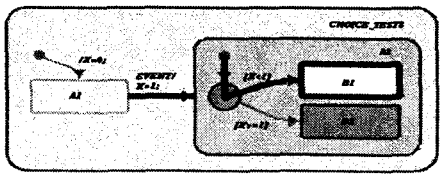

그림 10. STATEMATE statechart : static choice

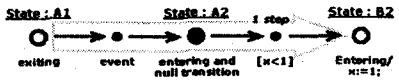

그림 11. STATEMATE statechart : static choice

3.2 우선순위와 스코프

스코프는 전이가 발생하는 원 상태와 목적 상태를 모두 포함하고 있는 가장 작은 상태 영역을 의미한다 [12].

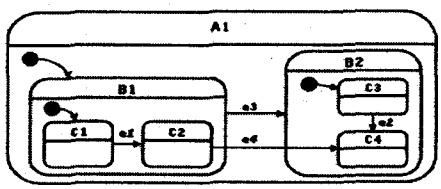

그림 12. Scope

그림 12에서 상태 C1이 C2로 전이될 때, 이들의 스코프는 C1과 C2를 모두 포함하는 가장 작은 상태, B1이 된다. 또한 상태 C2가 e4에 의해 C4로 전이될 때, 이들을 모두 포함하는 가장 작은 상태는 A1이므로 이 전이의 스코프는 상태 A1이 된다.

Statechart에서 전이의 충돌이 발생했을 때, 일반적으로 전이가 일어나는 스텝의 원 상태와 목적 상태의 스코프를 살펴보게 된다. Rhapsody와 STATEMATE의 의미론은 이들의 스코프에 따라 전이의 우선순위를 부여하는 방식이 각각 다르기 때문에, 이번 절에서는 3가지 시뮬레이션을 통해 이들의 의미론을 비교해 본다.

1) 서로 다른 스코프 상의 전이 우선순위 비교

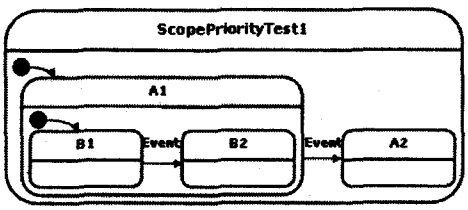

그림 13. Scope Priority Test 1 (UML)

그림 13은 각 전이의 원 상태와 목적 상태가 동일한 스코프에 존재하지만, 이들 전이들은 각각 스코프의 레벨이 다른 경우이다. 상태 B1이 B2로 전이되는 스코프는 A1이고, A1이 A2로 전이되는 스코프는 최상위 레벨이다. 동일한 event에 대해 서로 다른 스코프의 전이 결과에 따라 각 statehcart 별 스코프의 우선순위를 살펴볼 수 있다 [4].

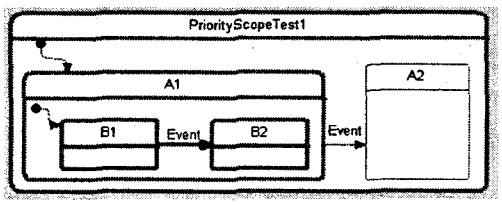

그림 14. Priority Scope Test 1 (Rhapsody)

Rhapsody statechart의 경우, PriorityScopeTest1이 시작되

면 초기 전이에 의해 그 하위의 A1과 B1이 각각 활성화 되는데, 이때 Event가 발생하면 B1이 B2로 전이된다. 그림 14의 시뮬레이션 결과에 의해 Rhapsody statechart 의미론은 낮은 level의 스코프에 priority가 주어진다는 것을 확인할 수 있다.

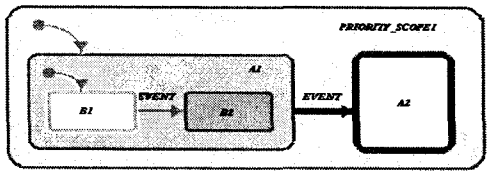

그림 15. Priority Scope Test 1 (STATEMATE)

하지만 그림 15를 통해 STATEMATE는 이들을 다른 방식으로 처리한다는 것을 볼 수 있다. PRIOIRITY_SCOPE1이 시작되어 A1과 B1이 각각 활성화 되었지만, EVENT에 의해 상위 스코프에 있는 트리거가 처리되어 A2로 전이되는 것을 알 수 있다.

위 예제 PriorityScopeTest1을 통해 Rhapsody의 의미론에서 동일한 event에 대한 스코프의 우선순위는 낮은 스코프의 트리거에 있는 반면에 STATEMATE 의미론은 높은 스코프에 우선순위가 주어진다는 것을 확인할 수 있다.

2) 원 상태와 목적 상태의 레벨이 각각 다른 전이 비교

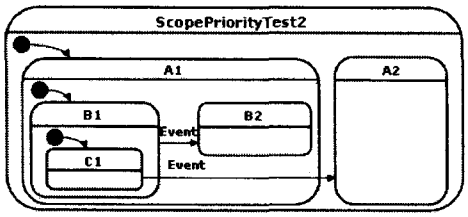

그림 16. Scope Priority Test 2

그림 16은 각 전이의 원 상태와 목적 상태가 각각 다른 레벨에 존재하며, 스코프도 각각 다른 경우이다. 앞선 예제에서 Rhapsody는 낮은 레벨의 스코프에, STATEMATE는 높은 레벨의 스코프에 각각 높은 우선순위를 부여한다는 것을 확인했기 때문에, 이번에는 각각 원 상태와 목적 상태의 레벨을 각각 달리하여 결과를 살펴본다. 우선 ScopePriorityTest2가 시작되면 그 내부의 상태 A1, B1, C1이 각각 활성화 되고, event가 발생했을 때 B1이 B2로 전이될 수도 있고 C1이 A2로 전이될 수 있는 우선순위의 충돌이 발생하는 상황이다.

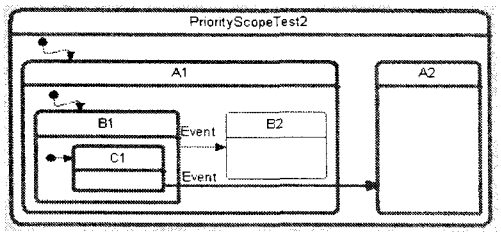

그림 17. Scope Priority Test 2 (Rhapsody)

Rhapsody는 C1이 A2로 전이되었다. Rhapsody가 낮은 레벨의 스코프에 우선순위를 부여함에도 불구하고 위와 같은 결과가 발생한 것은 전이가 발생하는 원 상태에 높은 우선순위가 있기 때문이다. 그림 14와 그림 17에 의해 Rhapsody statechart에서 원 상태가 서로 다른 레벨에 위치할 때, 전이의 충돌은 낮은 레벨의 원 상태를 갖는 전이에 높은 우선순위가 부여된다는 것을 확인할 수 있다.

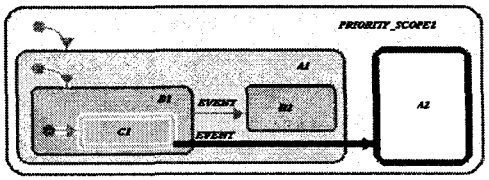

그림 18. Scope Priority Test 2

한편, STATEMATE는 원 상태와 목적 상태를 모두 포함하는 스코프를 보고 판단하며, 상위 스코프에 높은 우선순위를 부여한다. 상태 B1이 B2로 전이되는 스코프 A1 보다 상태 C1이 A2로 전이되는 최상위 레벨의 스코프에 높은 우선순위가 주어지기 때문에 상태 C1이 A2로 전이되는 것을 확인할 수 있다.

3) 다른 레벨의 원 상태에서 같은 레벨의 목적 상태간 전이 비교

앞선 두 가지 예제의 결과를 종합하기 위해 서로 다른 레벨의 원상태에서 동일한 레벨의 상태로 전이되는 경우를 확인해 보았다. 상태 A1이 A2로 전이되는 경우와 B1이 A3로 전이되는 경우 모두 이들의 스코프는 최상위 레벨이고, 목적 상태가 같은 레벨에 존재하고 있기 때문에 서로 다른 레벨의 원 상태에 따른 전이 결과를 확인해 볼 수 있다.

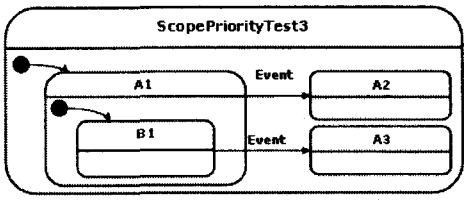

그림 19. Scope Priority Test 3 (UML)

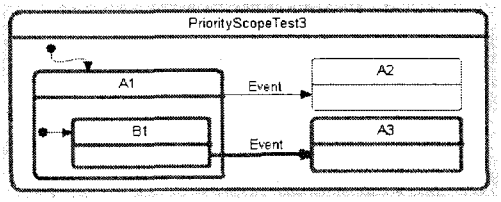

그림 20. Scope Priority Test 3 (Rhapsody)

Rhapsody는 앞선 예제의 결과와 같이 낮은 레벨의 원 상태에 높은 우선순위가 부여되기 때문에 활성화 된 최하위 상태 B1이 A3로 전이된다. 반면에 STATEMATE는 트리거의 스코프에 의해 전이가 결정되기 때문에 원 상태와 목적 상태를 모두 포함하는 가장 낮은 상태를 찾아야 한다. 이 경우에는 두 트리거 모두 스코프가 최상위 레벨의 상태로 같기 때문에 미결정상태(Non-determinism)에 빠져 있음을 확인할 수 있다.

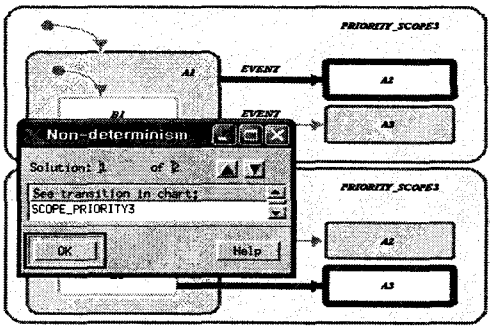

그림 21. Scope Priority Test 3 (STATEMATE)

3.3 복합 액션의 실행 순서

다음은 한 트리거 내에서 여러 action을 수행하는 경우에 이들에 대한 순서와 이들 action이 상호간 영향에 영향을 미칠 수 있는지의 여부에 대해 알아보기 위한 예제이다 [6, 10].

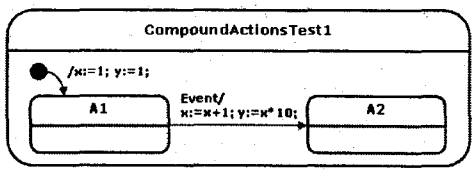

그림 22. Compound Actions Test1 (UML)

Rhapsody 의미론은 트리거의 action들을 순차적으로 수행하며, 이 과정에서 버퍼를 사용하지 않고 실제 정의된 변수에 각각 직접 기록하기 때문에 선행 action이 후행 action에 영향을 미칠 수 있다. 그림 23은 위 예제를 Rhapsody에서 시뮬레이션한 결과이다. 초기 전이로 변수 x와 y에 각각 1이 할당된 상태 A1에서 Event1이 발생하면, x에 1을 증가시키고, 변화된 x의 값을 이용해 y를 계산해 내기 때문에 아래와 같은 결과를 얻을 수 있다.

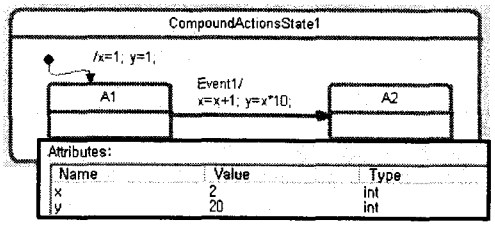

그림 23. Compound Actions Test1 (Rhapsody)

하지만 STATEMATE 의미론은 트리거의 action들을 병렬적으로 수행하며, 이 과정에서 내부적으로 정의된 버퍼 변수에 각각 임시 저장하기 때문에 선행 action이 후행 action에 영향을 미치지 않는다.

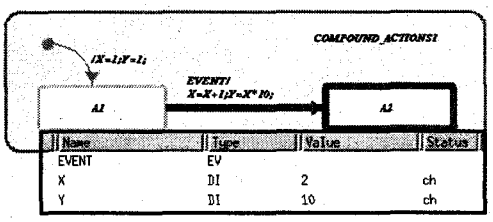

그림 24. Compound Actions Test1 (STATEMATE)

버퍼는 소스코드 상에서 cgSingleBuffer_*AcitivityName* 이라는 구조체에 해당 변수들을 선언하고, 계산 과정에서는 실제 변수에 기록하기 전에 버퍼 공간에 각각 임시적으로 저장하기 때문에 후행 action이 동적으로 선행 action의 결과에 영향받지 않는다.

3.4 동시 전이의 실행 순서

아래의 그림과 같은 AND-state는 복수개의 OR-state 영역으로 구성되어 있다. OR-state가 갖는 의미는 해당 영역에서의 제어가 유일하고 배타적으로 주어진다는 것이고, 각각의 독립된 영역에 제어가 모두 하나씩 존재해야 한다는 것이다. 다음은 AND-state에서 event에 의한 동시전이에 대한 예제이다 [6, 10].

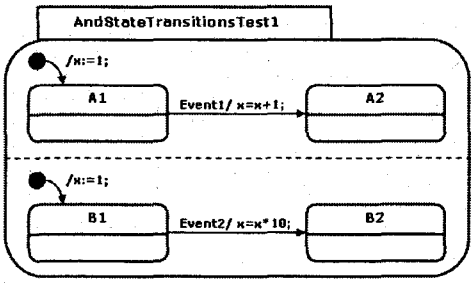

그림 25. And State Transitions Test1

Rhapsody에는 이벤트 큐가 있기 때문에 서로 다른 두 이벤트가 발생해도 이들을 각각 순차적으로 처리한다. 위 예제의 경우 AND-state에서 상태 A1과 B1이 각각 활성화 되어 있을 때, Event1과 Event2가 이벤트 큐에 들어가면 이들 이벤트는 진입 순서에 따라 각각 순차적으로 처리된다.

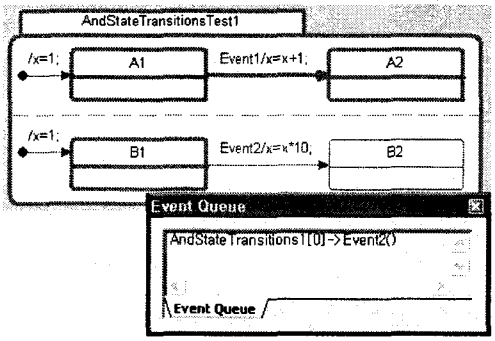

그림 26. And State Transitions Test1 (1)

따라서 그림 26과 같이, 하나의 이벤트가 먼저 처리되어 상태 A1에서 A2 혹은 상태 B1에서 B2로의 트리거 둘 중 하나가 먼저 전이되고, 나머지는 이벤트 큐에 남아있는 이벤트에 의해 나중에 처리된다. 즉, Rhapsody의 AND-state에서 동시전이는 순차적으로 처리되며, 이는 이벤트 큐에 들어간 이벤트의 순서에 의해 결정된다.

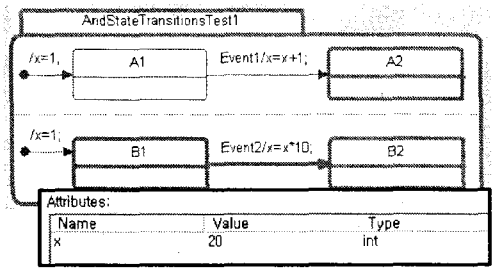

그림 27. And State Transitions Test1 (2)

그림 27은 위 예제에서 Event1과 Event2가 순서대로 이벤트 큐에 들어갔을 경우를 보여준다. x의 값은 (1+1)*10 이므로 20이 되고, event 순서가 반대의 경우 x의 값은 (1*10)+1 이므로 11이 된다.

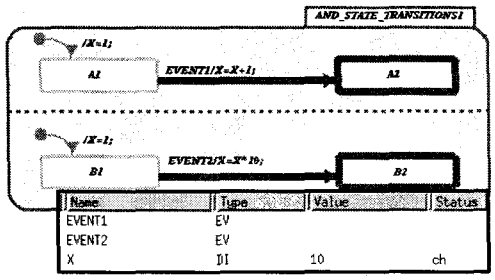

그림 28. And State Transitions Test1 (2)

STATEMATE는 전이가 완료된 후에 실제 변수에 값을 기록하며, 이 action들은 병렬적으로 처리되기 때문에 둘 중 한 가지는 무시된다고 볼 수 있다. 이러한 모델의 구문 혹은 의미론적인 에러와 경고를 검출하기 위해 STATEMATE 에서는 Model Checker와 Model Certifier를 사용하는데, 위의 그림 28과 같이 동일한 변수에 동시에 값을 배정해야 하는 경우, 이는 Model Checker를 통해 write/write race condition을 검출해 낼 수 있다.

4. 결론 및 향후 연구

Statechart는 시스템의 행위를 모델링하기 위한 명세 언어로서, 구조적 분석을 기반으로 처음 제안되었다. 하지만 소프트웨어 개발 패러다임이 변화하면서 statechart 역시 다양한 형태의 변형들이 생겨나게 되었다. Statechart를 개발 시스템에 적용하기 위해 시스템의 도메인에 맞는 적절한 statechart를 선택하는 것은 매우 중요하지만, 각 statechar와 이를 지원하는 자동화 도구의 사용에도 각기 다른 제한사항이 있기 때문에 이들 간 statechart의 상호 변환이 필요하다. 특정 statechart에 따라 대상 시스템을 명세하고, statechart의 상호 변환 과정을 거쳐 또다른 지원 도구를 통해 이를 검증하면 보다 완성도 높은 설계를 할 수 있다.

본 논문에서는 statechart에 따른 의미론적인 차이점을 살펴보기 위해 제시된 예제를 시뮬레이션하며 이들의 특징을 비교 분석하였다. 현존하는 statechart와 이에 대한 각 지원도구가 갖는 제약사항은 statechart 모델간 상호 변환을 통해 보완이 가능하다. 위의 비교 분석 내용을 기반으로 다양한 변환 알고리즘 개발하고 이를 통해 변환 규칙을 명세하면, 상호 변환을 위한 중간 단위의 표준화 된 언어를 정의하고 이에 따른 모델 변환 도구를 개발할 수 있기 때문에 이에 대한 연구가 필요하다.

5. 참고문헌

[1] Michael von der Beeck, "A Structured operational semantics for UML-statecharts", Software and Systems

[2] David Harel, "Some Thought on Statecharts, 13 Years Later", In Proceedings of the 9th International Conference Computer Aided Verification (CAV '97), Lecture Notes in Computer Science, vol. 1254, pp. 226-231, Springer, 1997

[3] David Harel, "On Visual formalisms", Communications of the ACM, volume 31, number 5, pp. 514-530, 1988

[4] Gerald Lüettgen, Michael von der Beeck and Rance Cleaveland, "A Compositional Approach to Statecharts semantics", Report 12, Institute for Computer Applications in Science and Engineering (ICASE 2000)

[5] David Harel, "Statecharts: A Visual formalism for Complex Systems", Science of Computer Programming, vol. 8, issue 3, pp. 231-274, 1987

[6] Michelle L. Crane, "On the Syntax and semantics of State Machines", PhD thesis, School of Computing Queen's University, Kingston, Ontario, Canada, Febrary 2006

[7] David Harel and Amnon Naamad, "The STATEMATE semantics of Statecharts", ACM Transactions on Software Engineering and Methodology (TOSEM), volume 5 , issue 4, pp. 293-333, 1996

[8] David Harel and Hillel Kugler, "The RHAPSODY semantics of Statecharts (or, On the Executable Core of the UML) - Preliminary Version", In Proceedings of the 3rd International Workshop on Integration of Software Specification Techniques for Applications in Engineering (INT 2004), Lecture Notes in Computer Science, vol. 3147, pp. 325-354, Springer-Verlag, 2004

[9] OMG, Unified Modeling Language (UML) 2.0 Superstructure Specification, Document formal/05-07-04, The Object Management Group (OMG), 2005

[10] Michelle L. Crane and Juergen Dingel, "UML vs. Classical vs. Rhapsody Statecharts: Not All Models are Created Equal", In Proceedings of ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2005), Lecture Notes in Computer Science, vol. 3713, pp. 97-112, Springer-Verlag, 2005

[11] OMG, Unified Modeling Language (UML) Specification 1.5 , Document formal/03-03-01, The Object Management Group (OMG), 2003

[12] Michael von der Beeck, "A Comparision of Statecharts Variants", In Proceedings of the 3rd International Symposium Organized Jointly with the Working Group Provably Correct Systems (ProCoS, 1994), Lecture Notes in Computer Science, vol. 863, pp. 128-148, Springer-Verlag, 1994Modeling (SoSyM), volume 1, number 2, pp. 130-141, Springer, 2002