DEVSimHLA를 이용한 시뮬레이터 연동

김재현, 김탁곤
한국과학기술원 전자전산학과
대전광역시 유성구 구성동 373-1
전화 : 042-869-5454
E-mail: {jhkim@smslab.kaist.ac.kr, tkim@ee.kaist.ac.kr}

Simulator Interoperation Using DEVSimHLA

Jae-Hyun Kim and Tag Gon Kim
Department of EE&CS, KAIST
373-1 Kuseong-dong Yuseong-ku, Daejeon

요 약

High Level Architecture (HLA)는 이기종 시뮬레이터 간의 연동을 위한 미들웨어이다. 본 논문에서는 Discrete Event System Specification (DEVS) 형식론을 바탕으로 한 DEVS 모델의 시뮬레이션을 HLA 기반에서 수행하는 방법에 대하여 제시한다. DEVS 시뮬레이션 메시지를 HLA의 서비스를 사용하여 변환하여 기존의 시뮬레이션 알고리즘을 그대로 사용하여 분산 시뮬레이션 할 수 있는 방식에 대하여 기술한다. 또한 모델 사이의 연결 정보를 HLA 환경에서 이용하는 방식에 대하여 기술한다. 제안된 방식을 구현한 DEVSimHLA 시뮬레이션 환경의 구조 및 동작 방식에 대하여 알아보고 이를 이용한 간단한 예제를 보인다.

I.서 론

High Level Architecture (HLA)는 상위 추상 레벨에서 데이터 모델과 인터페이스를 표준화함으로써 시뮬레이션 분야에서 재사용성을 높이고 이기종 시뮬레이션 간의 연동을 가능하게 하였다. HLA가 시뮬레이션 레벨에서의 연동을 지원하지만, 각 시뮬레이션 자체의 개발은 개발자의 담당이다. 효율적인 시뮬레이션의 개발을 위해서는 모델링/시뮬레이션을 위한 틀이 필요하고, 그 틀의 하나로 Discrete Event System Specification (DEVS) 형식론이 있다. DEVS 형식론을 바탕으로 하여 HLA를 사용하는 시뮬레이션을 개발할 수 있는 환경이 있다면 개발자가 쉽고 편리하게 분산 시뮬레이션을 수행할 수 있다. 본 논문에서는 HLA환경에서 DEVS 모델을 시뮬레이션 하는 방법에 대하여 제시한다.

II.DEVS 형식론 및 시뮬레이션 알고리즘

가.DEVS 형식론

DEVS 형식론은 집합 이론에 근거하여, 이산 사건 시스템을 모델링 할 수 있는 방법을 제시한다.

DEVS 형식론에는 원자모델(Atomic Model)과 결합 모델(Coupled Model)의 두 가지 모델 종류가 있어 시스템을 계층적이고 모듈러한 방식으로 모델링할 수 있도록 한다.

원자 모델은 시스템의 가장 작은 구성요소를 나타내는 모델로, 시간 명세 상태 천이 (Timed State Transition) 레벨에서 시스템의 동작을 표현한다. 원자 모델의 정의는 다음과 같다.

$$AM = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

X : 입력사건집합
S : 상태집합
Y : 출력사건집합
δ_{int} : $S \rightarrow S$: 내부상태전이함수
δ_{ext} : $Q \times X \rightarrow S$: 외부상태전이함수
λ : $S \rightarrow Y$: 출력함수
ta : $S \rightarrow Real$: 시간전진함수
$Q = \{(s, e) | s \in S, 0 \le e \le ta(s)\}$
: total state of AM(e: elapsed time)

결합 모델은 원자 모델이나 또 다른 결합 모델로 기술된 구성 요소 모델들을 연결하여 만든 상위 모델이다. 결합 모델의 정의는 다음과 같다.

$$CM = \langle X, Y, \{M_i\}, EIC, EOC, IC, SELECT \rangle$$

X : 입력사건집합
Y : 출력사건집합
$\{M_i\}$: DEVS컴포넌트집합
$EIC \subseteq X \times \bigcup_i X_i$: 외부입력관계
$EOC \subseteq \bigcup_i Y_i \times Y$: 외부출력관계
$IC \subseteq \bigcup_i Y_i \times \bigcup_i X_i$: 내부입출력관계
$SELECT$: $2^{\{M_i\}} - \phi \rightarrow \{M_i\}$

나.추상화된 시뮬레이터

DEVS 형식론으로 기술된 모델은 그림 1의 왼편에 표현된 것처럼 계층적 트리 구조를 갖는다. 트리 구조의 각 단말에는 원자 모델이 위치하고, 중간에는 결합모델이 위치한다.

DEVS 모델을 시뮬레이션 하기 위한 시뮬레이터의 구조도 이와 동일하다. 원자 모델에는 Simulator, 결합 모델에는 Coordinator라고 하는 프로세스가 있어서 적절한 알고리즘에 따라 해당 모델의 특성 함수를 실행하여 시뮬레이션을 수행한다.

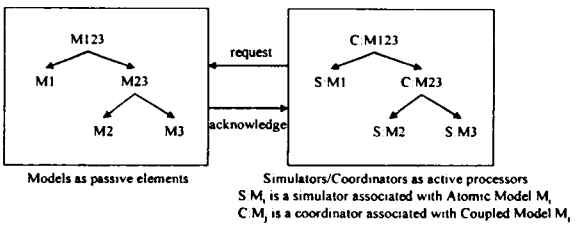
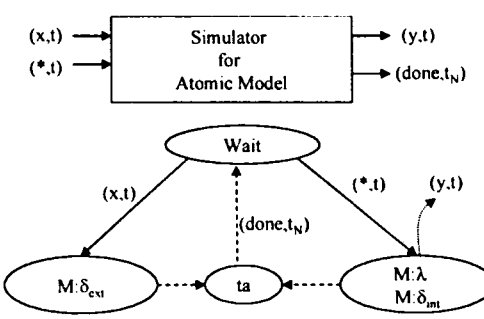


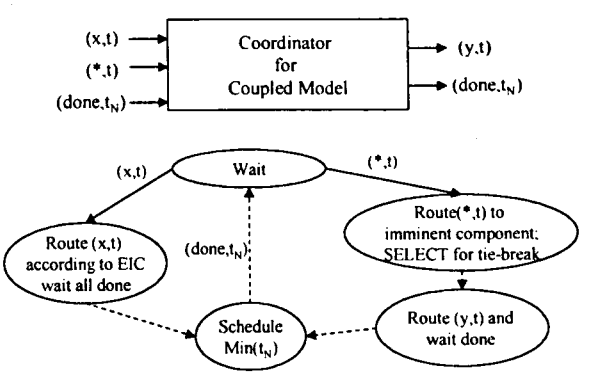
그림 1 추상화된 시뮬레이터의 구조

표 2 시뮬레이션 메시지

메시지	의미
(x,t)	시간 t에서의 외부 입력 이벤트
(y,t)	시간 t에서 생성된 출력 이벤트
(*,t)	스케줄된 시간 t가 되었음을 나타내는 내부 이벤트
(done,t_N)	다음 스케줄이 t_N이라는 것을 알리는 메시지



(a) Simulation Algorithm for Simulator



(b) Simulation Algorithm for Coordinator

그림 2 시뮬레이션 알고리즘

앞의 표1과 그림2는 각각 시뮬레이션에 사용되는 메시지의 종류와 시뮬레이션 알고리즘을 나타낸다.

원자 모델은 발생된 이벤트에 따라 상태 천이가

일어나도록 되어 있다. 이벤트의 종류는 외부에서 들어오는 입력 이벤트 (x,t)와 내부적으로 발생한 내부 이벤트 (y,t)가 있다. 원자 모델이 머물고 있는 한 상태(State)에는 그 상태에 머물 수 있는 시간이 명시되어 있고, 그 시간이 되면 내부 이벤트가 발생하여 상태 천이를 하도록 되어 있다. 만약 그 시간이 지나기 전에 외부 입력이 들어오면 발생된 이벤트에 따라 상태가 바뀐다.

결합 모델은 내부 구성요소 모델에서 발생한 이벤트 혹은 상위 결합 모델에서 보내준 메시지를 내부 구성요소 모델로 전달하는 역할을 한다. 또한 내부 모델의 다음 이벤트 시간들을 모아서 그 중 가장 작은 값을 상위 결합 모델에 보고하는 역할을 한다. 결국 가장 상위에 위치한 결합 모델은 전체 모델의 다음 이벤트 시간 중 가장 빠른 시간을 알게 되고, 그 시간으로 전체 시뮬레이션 시간을 진행시킨 후, (*,t) 메시지를 발생시킨다.

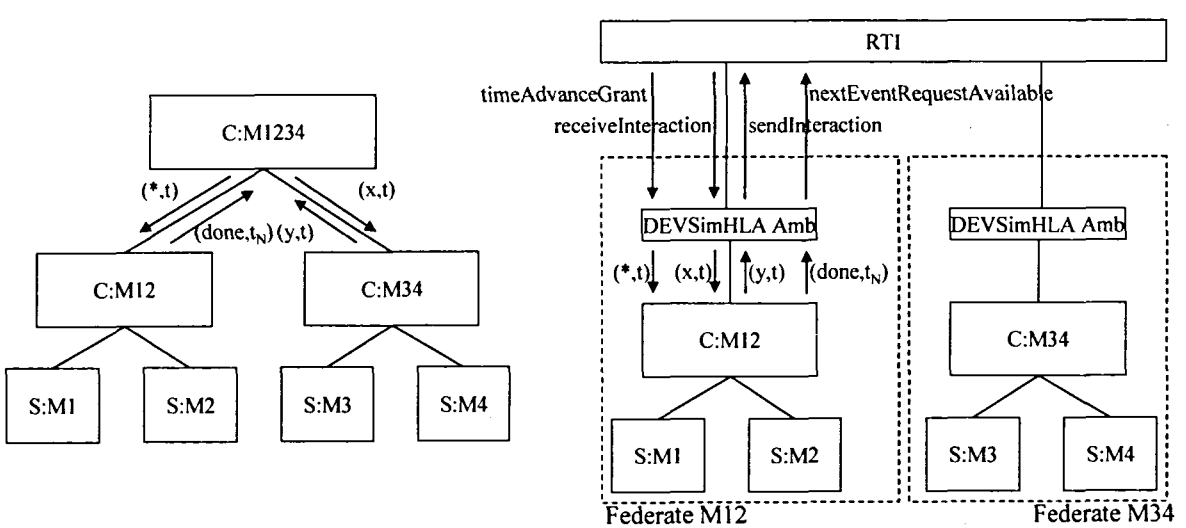
III.DEVSimHLA

앞서 설명한 DEVS의 추상화 시뮬레이션 알고리즘을 객체지향형 언어인 C++로 구현한 것이 DEVSim++ 시뮬레이션 환경이다.[?] DEVSim++ 환경은 원자 모델, 결합 모델을 기술할 수 있는 클래스를 포함하여 모델링 및 시뮬레이션에 필요한 제반 클래스들을 포함하고 있는 라이브러리이다.

DEVSimHLA는 DEVSim++을 확장하여 HLA/RTI를 사용할 수 있도록 한 것이다. HLA/RTI는 분산 시뮬레이션에서 각 시뮬레이터들의 시간 관리 및 메시지 교환 등을 편리하게 할 수 있도록 도와주는 미들웨어이다. 따라서 DEVSimHLA는 기존의 Root Coordinator 혹은 최상위 결합 모델의 Coordinator에서 해 오던 전체적인 시간 관리를 RTI에게 맡기는 형식으로 디자인되었다.

다음 그림 3은 순차적 시뮬레이터인 DEVSim++와 DEVSimHLA의 차이를 보여준다. M12, M34라는 결합 모델이 이미 개발되어 있다고 하자. 이를 재사용하여 M1234라는 모델을 만들어 시뮬레이션을 수행하고자 할 때 그림 3의 (a)와 같은 구조를 가진다. 최상위 결합모델 M1234의 Coordinator는 하위 모델들 중 스케줄된 이벤트가 가장 빠른 모델에게 (*,t)를 발생시키는 시간관리 기능과, 하위 모델에서 발생되는 (y,t)를 (x,t)로 바꾸어 적절히 전달하는 메시지 교환의 역할을 한다.

DEVSimHLA에서는 이 두 역할을 RTI가 대신한다. 그림 3의 (b)는 DEVSimHLA를 이용하여 모델을 구성한 예를 보인다. 새로운 모델 M1234를



(a) Sequential Simulation **(b) DEVSimHLA**

그림 3 DEVSimHLA의 구조와 메시지 변환

만드는 것 대신에 기존의 M12, M34 모델을 DEVSimHLA를 사용하여 각각의 Federate로 만든다. DEVS 형식론의 장점 중 하나가 모델과 시뮬레이터의 분리이다. 따라서 기존에 DEVSim++ 환경에서 개발된 M12, M34 모델은 소스 코드 레벨에서 수정할 사항이 전혀 없이 DEVSimHLA 환경에서 사용가능하다.

M12, M34 모델의 Coordinator는 RTI와 통신을 하는 DEVSimHLA Ambassador와 연결이 된다. 이 Ambassador에서 시뮬레이션 메시지의 변환이 일어난다. Ambassador는 Coordinator에서 보내는 시뮬레이션 메시지를 받아서 RTI의 함수를 호출하거나, RTI로부터의 결과를 받아서 Coordinator에게 전달하는 역할을 한다.

먼저 시간 관련 메시지인 (*,t)와 (done,t_N)에 대해서 알아보자. (done,t_N)은 하위모델에서의 이벤트 처리를 끝마치고 t_N까지는 이벤트를 발생시키지 않기 때문에 t_N까지 시간 진행을 하도록 요청하는 것이다. 따라서 (done,t_N) 메시지는 RTI에서 시간 진행을 요구하는 함수를 사용하여 변환한다.

RTI에서 시간진행을 요청하는 함수는 여러 종류가 있다. 지정된 시간까지 시간 진행을 요구하는 timeAdvanceRequest/timeAdvanceRequestAvailable 함수들과, 지정된 시간 혹은 다음 이벤트가 발생하는 시간까지 시간 진행을 요구하는 nextEventRequest / nextEventRequestAvailable 함수들이 있다. DEVS의 시뮬레이션 알고리즘에서는 한 모델이 (done,t_N)을 발생하고 시간 진행을 기다리고 있을 때 외부에서 (x,t)가 발생할 수 있다. 이 (x,t)를 받으면 상태 천이를 하고 이에 따른 스케줄을 다시 수행하게 된다. 따라서 다음 이벤트가 발생할 때까지만 시간 진행을 해야 하므로 nextEventRequest 류의 함수를 사용해야 한다.

DEVS 형식론에서는 출력이벤트(y,t)가 발생한 순간 같은 시간에 입력이벤트(x,t)로 들어가도록 되어 있다. 이러한 특성 때문에 분산 환경에서 Conservative 방식으로 시간 관리를 구현할 때 DEVS 시뮬레이터는 zero-Lookahead를 가져야 한다. 따라서 zero-Lookahead를 지원하는 nextEventRequestAvailable 함수를 사용하여 (done,t_N) 메시지를 변환한다.

이렇게 시간 진행 요청을 하였을 때 그 만큼 시간 진행을 해도 좋다는 것이 timeAdvanceGrant Callback 함수이다. 이 함수가 불리면, 인자로 지정된 시간까지 시간 진행을 해도 된다는 의미이다. 따라서 시간 요청을 했을 때의 시간과 허락 받은 시간이 같으면 (*,t)를 발생시킨다. 허락 받은 시간이 요청한 시간보다 작은 경우는 그 시간에 외부에서 입력이벤트가 발생한 경우로써, 새로운 스케줄을 위해 (done,t_N) 이벤트를 발생시키게 되고 t_N 값으로 시간 진행 요청을 다시 하게 된다.

다음은 모델끼리 주고 받는 메시지의 변환에 대해 알아보자. M12의 하위모델인 M1이 M34의 하위 모델인 M3로 메시지를 보낸다고 가정하자. C:M12는 M1->M2 메시지는 내부적으로 전달하지만, M3인 경우는 자신에 속한 모델이 아니므로 상위 Coordinator로 전달하게 된다. 이 경우 상위 Coordinator는 RTI가 된다.

RTI의 서비스 중 Federate 사이의 메시지를 주고 받는 것을 Interaction이라고 하고, sendInteraction / receiveInteraction Callback 서비스를 제공한다. 출력이벤트(y,t)는 sendInteraction 함수를 호출하는 것으로 변환한다. 메시지를 받는 Federate는 receiveInteraction Callback 함수가 RTI에 의해 불리게 된다. 이 Callback함수내에서 (x,t)를 발생시키게 되고 하위 Coordinator로 전달하게 된다.

모델 M12와 M34가 어떻게 연결되는 지는 다음과 같은 스크립트의 형태로 작성하여 각 Federate에서 초기화시 읽어서 저장하여 사용한다.

```
Federation M1234;
```

```
component
{
        M12, M34
}

coupling
{
        M12.out->M34.in
}
```

IV.방공 모델 예제

위의 그림 4는 간단한 방공 모델 예제이다. 기존에 개발되어 있는 지상 레이더와 미사일 발사 시스템 모델(Land)과, 발사된 미사일과 표적 항공기의 궤적을 계산하는 모델(Air)을 연결하여 시뮬레이션을 수행한다. 다음 그림 5는 DEVSimHLA를 이용하여 시뮬레이션을 수행하는 화면이다.

V.결론 및 추후과제

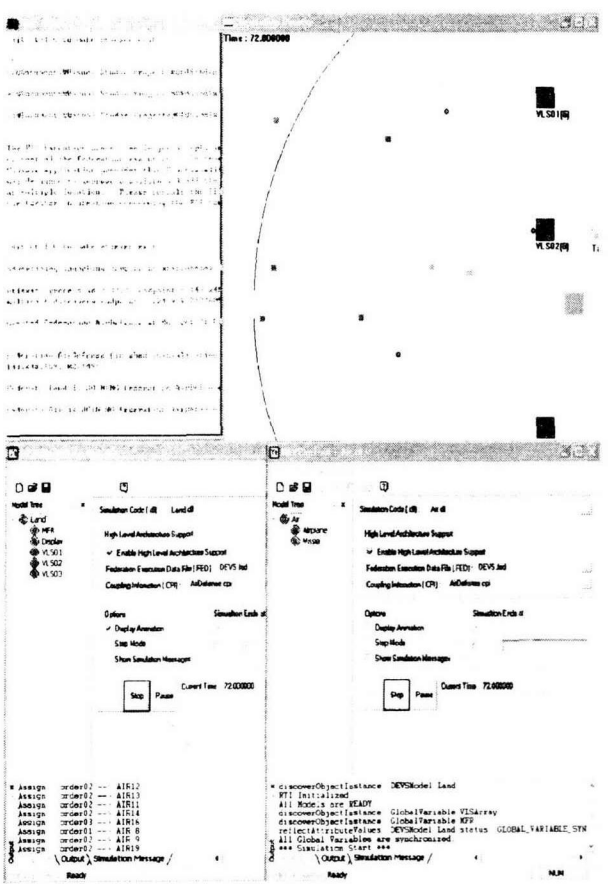


그림 4 DEVSimHLA 화면

본 논문에서는 DEVS 모델을 HLA 환경에서 시뮬레이션 하기 위한 환경인 DEVSimHLA에 대해서 설명하였다. DEVS의 시뮬레이션 메시지를 HLA에서 제공하는 인터페이스에 맞추어 변환하는 방법과 모델 사이의 연결 정보를 입력하는 방법을 제시하였다.

현재 제시한 방법은 DEVS의 순차적 시뮬레이션 알고리즘을 수정하지 않고 HLA/RTI에서 제공하는 서비스를 사용하는 방법이다. 따라서 zero Lookahead를 사용하였는데, Lookahead값이 작을수록 분산 시뮬레이션의 성능은 떨어지게 된다.

추후 과제로써 DEVS 형식론에서 zero Lookahead를 사용하지 않고 Conservative 방식의 분산 시뮬레이션 알고리즘을 고안할 필요가 있다.

참 고 문 헌

[1] B.P.Zeigler, H. Praehofer and T.G. Kim, *Theory of Modeling and Simulation*, 2nd ed., Academic Press, 2000

[2] IEEE Std 1516-2000, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture(HLA) - Framework and Rules.

[3] IEEE Std 1516.1-2000, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture(HLA) - Federate Interface Specification.

[4] IEEE Std 1516.2-2000, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture(HLA) - Object Model Template (OMT) Specification.