# DEVS-HLA: 이 기종 분산 시뮬레이션 틀

## DEVS-HLA: Distributed Heterogeneous Simulation Framework

김용재*, 김탁곤*
Yong Jae Kim and Tag Gon Kim

**Abstract**

We describe a heterogeneous simulation framework, so called DEVS-HLA, in which conventional simulation models and the DEVS (Discrete Event System Specification) models are interoperable. DEVS-HLA conceptually consists of three layers: model layer, DEVS BUS layer, and HLA (High Level Architecture) layer. The model layer has a collection of heterogeneous simulation models, such as DEVS, CSIM, SLAM, and so on, to represent various aspects of a complex system. The DEVS BUS layer provides a virtual software bus, DEVS BUS, so that such simulation models can communicate with each other. Finally, the HLA layer is employed as a communication infrastructure, which supports several good features for distributed simulation. The DEVS BUS has been implemented on the HLA/RTI (Run-Time Infrastructure) and a simple example of a flexible manufacturing system has been developed to validate the DEVS-HLA.

* 한국과학기술원 전자전산학과

## 1. 서 론

시뮬레이션은 대상 시스템의 특성을 분석, 평가하기 위해 사용된다. 특히, 대상 시스템이 존재하지 않는 설계 단계에서는 필수적인 도구이다. 시뮬레이션 모델은 시뮬레이션 목적에 맞게 대상 시스템을 잘 표현하여야 한다. 이때, 대상 시스템의 특성에 따라 여러 가지 모델 기술 방법들이 사용되어 왔다. 그러나, 대상 시스템이 복잡해질수록 시스템은 여러 가지 복합적인 특성들을 갖게 되고, 단일 모델 기술 방법으로는 시스템의 여러 가지 특성을 효율적으로 나타내기 어렵게 되었다. 또한, 분석 및 평가하고자 하는 문제 자체도 여러 가지 측면들을 포함하게 되었다. 이렇게 복합적인 시스템과 다측면의 문제를 해결하기 위해서, 시스템과 문제의 각 측면들에 대해 잘 기술된 다양한 모델들이 필요하다. 즉, 여러 가지 방법으로 기술된 모델들이 동시에 시뮬레이션 되어서 복잡한 문제에 대한 해답을 주어야 한다는 것이다.

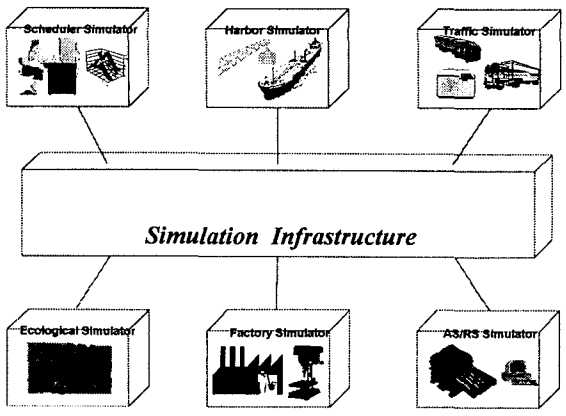
이러한 이 기종 모델들의 시뮬레이션 시에는 다양한 형태의 이 기종 시뮬레이션 환경들이 포함되므로, 전체 시뮬레이션 환경을 구축할 때 시뮬레이션 속도 향상보다는 상호 운용성(interoperability)에 더 중점을 두어야 한다. 최근 미 국방성에 의해 제안된 HLA (High Level Architecture)는 다양한 시뮬레이션 모델간의 상호 운용성을 높이기 위한 방법이다 [4]. HLA는 국방 분야의 모델링 및 시뮬레이션에서 각광을 받고 있으며, 일반 산업분야에서도 적용하고자 하는 노력이 이루어져왔다 [13, 16]. 그러나, 기존의 시뮬레이션 환경이 HLA를 통해 상호 연동되기 위해선 상당한 수정이 요구되어진다.

DEVS-HLA는 기존 시뮬레이터의 수정을 최소화 하는 것을 목표로 하는 이 기종 시뮬레이션 틀이다 [1, 8, 9, 10]. 노드 시뮬레이터들의 상호 연동을 위해 가상 소프트웨어 버스인 DEVS 버스를 사용하고, 시뮬레이션 프로토콜 변환을 통해 기존 시뮬레이션 환경이 DEVS 버스에 간단히 연동될 수 있도록 한다.

본 논문에서는 DEVS-HLA의 구조와 DEVS 버스 프로토콜 및 기존 시뮬레이터의 프로토콜 변환, DEVS-HLA의 구현, 그리고 DEVS-HLA의 간단한 적용 사례등을 기술한다.

## 2. 이 기종 시뮬레이션

이 기종 시뮬레이션은 서로 다른 시뮬레이션 언어로 기술된 모델들이 포함된 시뮬레이션이다. 대상 시스템이 점점 복잡해짐에 따라 여러 가지 측면을 동시에 고려해야 하고, 이에 따라 이 기종 시뮬레이션을 수행해야 할 필요성도 점점 증가하고 있다. 예를 들면, <그림 1>에 나타냈듯이 어떤 제조 회사의 시뮬레이션의 경우, 물류 시뮬레이터, 항만 시뮬레이터, 공장 시뮬레이터, 자동 창고 시뮬레이터, 일정 조정 시뮬레이터, 환경 시뮬레이터등의 여러 가지 시뮬레이터가 있어서, 일정 조정에 따른 물류 비용의 변화, 생산 시간의 단축 여부, 환경에 대한 영향등을 동시에 고려할 필요가 있을 수 있다. 이때, 이러한 이 기종 시뮬레이터들을 통합하기 위해 시뮬레이션 하부구조(infrastructure)가 필요하다. 시뮬레이션 하부구조는 이 기종 시뮬레이터들간의 데이터 교환 및 전체 시뮬레이션 시간 진행에 대한 동기 방법 등을 제공할 수 있어야 한다.

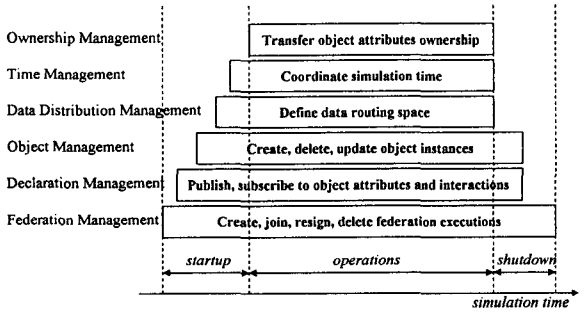



<그림 1> 제조회사의 이 기종 시뮬레이션

## 2.1 HLA

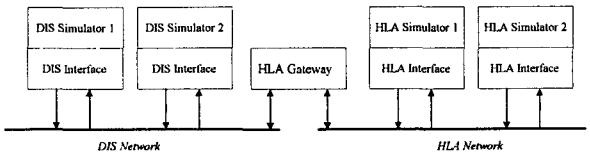
최근 미 국방성에 의해 제안된 HLA는 이 기종 시뮬레이션 모델간의 상호 운용성을 높이기 위한 방법이다 [4]. HLA는 크게 Rule, Interface Specification, 그리고 OMT (Object Model Template)로 정의된다. Rule은 노드 시뮬레이터와 시뮬레이션사이의 약속을 나타낸다. Interface Specification은 노드 시뮬레이터들과 RTI (Run-Time Infrastructure) 사이의 상호작용을 나타낸다. 마지막으로 OMT는 시뮬레이션 객체 정보를 표현하는 공통 방법을 나타낸다.

RTI는 노드 시뮬레이터에게 다음의 6가지 종류의 관리 체계를 제공한다. 즉, federation management, declaration management, object management, ownership management, time management, data distribution management이다. <그림 2>에는 각 관리 체계에서 제공하는 대표적인 기능들을 간략히 나타내었다 (자세한 것은 [6] 참조).




<그림 2> RTI의 6가지 관리 체계

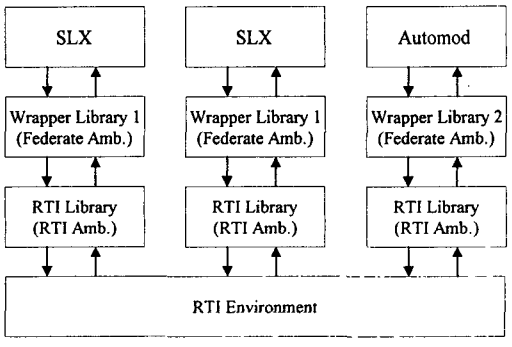
## 2.2 관련연구




<그림 3> HLA Gateway

HLA를 이용한 이 기종 시뮬레이션은 HLA Gateway [3] 와 Wrapper 방법 [13]등이 제안되었다. HLA Gateway는 기존의 DIS (Distributed Interactive Simulation) 프로토콜을 사용하는 시뮬레이션 모델과 HLA를 사용하는 모델을 연동시키기 위한 것으로, 두 개의 시뮬레이션 네트웍을 HLA Gateway로 연결한 것이다 (<그림 3>). 이때, HLA Gateway는 DIS의 패킷과 HLA의 RTI 서비스를 변환하는 일을 담당한다. 이 방법은 기존의 DIS 네트웍을 수정하지 않고 간단하게 HLA로 전이할 수 있는 방법이지만, 원래 분산 환경이 아닌 단일 시뮬레이션 환경들의 통합에는 부적절한 방법이다.

<그림 4>의 Wrapper 방법은 HLA를 산업계에서 이용하기 위한 시도로 기존의 시뮬레이션 환경인 SLX와 Automod등에 HLA를 위한 Wrapper 라이브러리를 제공한 것이다 [13]. 이것은 HLA/RTI에서 제공하는 모든 기능을 SLX나 Automod에서 실행할 수 있는 함수로 바꿔놓은 것이다. 한편, Wrapper 방법은 노드 시뮬레이터가 바뀌게 되면 Wrapper 라이브러리를 모두 수정해야 하는 문제가 생긴다. 반면, 본 논문에서 기술하는 DEVS-HLA는 이때 필요한 수정을 최소화하기 위한 방법이다.




<그림 4> Wrapper 방법

## 3. 이 기종 시뮬레이션 틀: DEVS-HLA

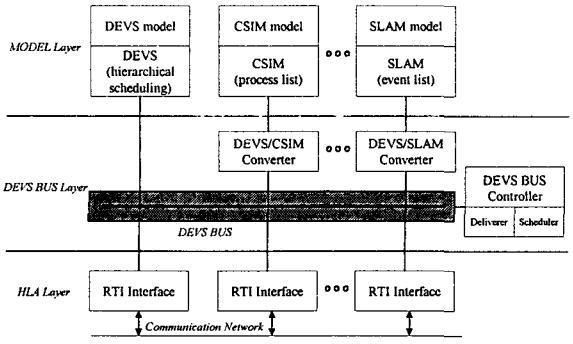
### 3.1 DEVS-HLA 구조

DEVS-HLA는 크게 모델 계층, DEVS 버스 계층, 그리고 HLA 계층으로 이루어진다 (<그림 5>). 각 시뮬레이션 노드는 시뮬레이션 모델, 시뮬레이션 프포토콜 변환기, 그리고 HLA/RTI 라이브러리로 구성된다.

모델 계층에서는 복잡한 대형 시스템을 여러 가지 형태의 모델 기술 방법들로 나타낸다. 예를 들면, DEVS, CSIM, SLAM등의 모델 기술 방법들을 사용할 수 있다. 여기에는 두 가지 장점이 있다. 첫째는 복잡한 시스템을 여러 가지 모델들을 사용하여 잘 표현할 수 있다. 둘째로는 기존에 잘 개발된 모델들을 재사용할 수 있다.

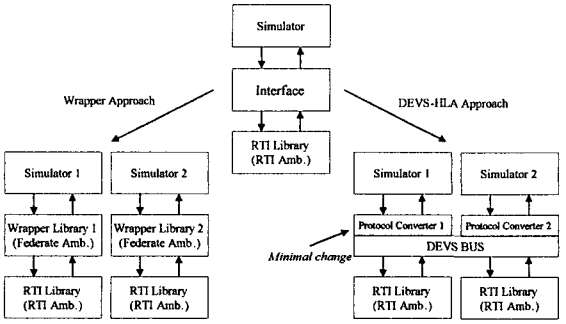
DEVS 버스 계층에서는 모델 계층에서 기술된 여러 가지 모델들이 DEVS 버스 프로토콜로 상호 연결된다. 이때, DEVS 모델을 제외한 다른 형태의 모델들의 경우에는 모델 해석 방법인 시뮬레이션 프로토콜이 다르기 때문에 시뮬레이션 프로토콜 불일치(mismatch)가 발생하므로 이를 해결하기 위해 프로토콜 변환기가 필요하다. 예를 들어, CSIM 시뮬레이션 모델의 경우, DEVS/CSIM 변환기는 DEVS 버스 메시지를 CSIM 프로세스로 전달하고, CSIM 출력 사건을 DEVS 버스 메시지로 변환한다. DEVS 버스 프로토콜이 제대로 동작하기 위해서 버스 제어기가 필요하다. 버스 제어기는 Deliverer와 Scheduler로 구성되어 있다. Deliverer는 노드 시뮬레이터들간의 데이터 교환을 위한 것이고, Scheduler는 DEVS 버스의 사용을 위한 시간 동기를 위한 것이다. 이들의 자세한 동작은 뒤에서 설명한다.

마지막으로, HLA 계층은 DEVS 버스를 구현하기 위한 통신 수단으로 사용된다. 기존의 분산 시뮬레이션을 위해서는 PVM이나 MPI등의 메시지 전달 도구들이 많이 사용되었는데, HLA는 이들에 비해 분산 시뮬레이션을 위한 여러 가지 유용한 기능들을 제공하므로 이 기종 시뮬레이션의 통신 수단으로 효과적으로 사용될 수 있다. 예를 들어, HLA는 분산 시뮬레이션에서 사용되는 보수적인 동기 방법(conservative synchronization)의 전체 시뮬레이션 시간 진행 방법을 제공해준다. 또한, Live Player나 운영자와의 상호작용(human interaction)등도 시뮬레이션에 포함될 수 있다.




<그림 5> DEVS-HLA 구조

DEVS-HLA와 Wrapper 방법과의 차이를 <그림 6>에 나타내었다. Wrapper 방법에서는 노드 시뮬레이터가 바뀜에 따라 새로운 Wrapper 라이브러리가 필요하다. 그러나, DEVS-HLA에서는 DEVS 버스라는 공통 부분을 재사용하고, 각 노드 시뮬레이터에서는 필요한 부분만 프로토콜 변환기를 통해 구현하므로, 노드 시뮬레이터에 요구되는 수정을 줄일 수 있다.




<그림 6> DEVS-HLA와 Wrapper 방법의 비교

### 3.2 DEVS 형식론

Zeigler [17]에 의해 제안된 DEVS 형식론은 이산사건 시스템의 명세를 위한 도구로 컴퓨터 하드웨어 및 소프트웨어, 통신 시스템, 제조 시스템등의 여러 가지 시스템들의 모델링 및 시뮬레이션을 위해 사용되어 왔다. 특히, 모델의 행위와 구조를 구별하여 명세함으로써 계층적이고 구조화된 방법으로 복잡한 모델을 쉽게 구성할 수 있다.

DEVS 형식론은 크게 기본 (atomic) 모델과 결합 (coupled) 모델의 두 가지 모델로 나뉜다. 기본 모델은 시스템의 동적 특성을 나타내기 위해 사용되며 다음과 같이 정의된다.

$AM=\langle X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta\rangle$
$X$: input events set;
$Y$: output events set;
$S$: sequential states set;
$\delta_{ext}: Q\times X\rightarrow S$, external transition function;
$Q=\{(s, e)|s\in S, 0\leq e\leq ta(s)\}$: total state of $AM$;
$\delta_{int}: S\rightarrow S$, internal transition function;
$\lambda: S\rightarrow Y$, output function;
$ta: S\rightarrow Real_{0^+}^{\infty}$, time advance function;

X, Y는 기본 모델의 입력과 출력 사건 집합이고, S는 상태 변수 집합이다. $\delta_{ext}$는 외부 입력 사건에 대한 상태변화를 나타내고, $\delta_{int}$는 외부 입력 사건이 없이 자발적인 상태변화를 나타낸다. $\lambda$는 외부로의 출력을 어떻게 생성하는 가를 나타내고, $ta$는 한 상태에서 머무르는 시간을 기술한다.

결합 모델은 여러 기본 모델들을 하나의 통합된 모델로 관리하여, 복잡한 시스템을 효율적으로 명세하기 위한 방법이다. 이때, 각 기본 모델간의 연결과 우선 순위에 대한 기술이 필요하다. 즉, 다음과 같은 방법으로 나타낸다.

$CM=\langle X, Y, M, EIC, EOC, IC, SELECT\rangle$
$X$: input events set;
$Y$: output events set;
$M$: a set of components;
$EIC$: external input coupling relation;
$EOC$: external output coupling relation;
$IC$: internal coupling relation;
$SELECT: 2^M-\varnothing\rightarrow$M, tie-breaking selector function

X와 Y는 통합된 결합 모델의 입력, 출력 사건 집합이고, EIC와 EOC를 통해 내부 모델들로 전달된다. 즉, 외부 입력 사건은 EIC를 통해 연결된 모델 $M_i$로 전달되고, $M_j$에 의해 발생된 출력 사건은 EOC를 통해 외부로 전달된다. 내부 모델들간의 연결은 IC를 통해 이루어진다. SELECT는 동일한 시뮬레이션 시간에 여러 내부 모델들이 동시에 수행되어야 할 필요가 있을 때, 각 모델들의 우선 순위를 정해주기 위한 방법이다.

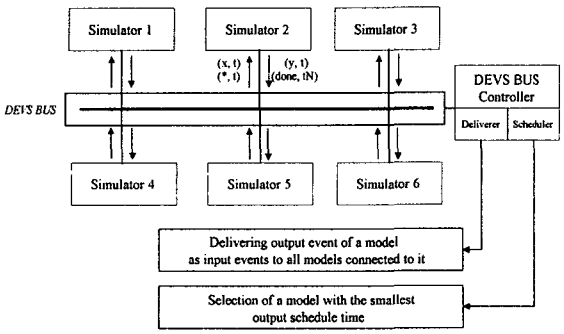
DEVS 형식론은 입력 사건과 출력 사건이 구분이 되고, 상태 천이 함수가 외부 사건에 의한 것과 내부의 시간 전진에 의한 것의 두 가지 형태가 있으며, 각 상태에서의 시간 전진을 기술함으로써 다양한 형태의 시간을 포함하는 이산 사건 시스템을 기술할 수 있다.

위의 두 가지 형태의 DEVS 모델로 잘 기술된 시스템을 시뮬레이션 하기 위해 추상화된 시뮬레이션 알고리즘이 사용된다 [17]. 기본 모델과 결합 모델 각각에 대응되는 시뮬레이션 알고리즘은 (x, t), (y, t), (*, t), (done, tN)의 네 가지 메시지를 사용한다. 이중, (x, t)와 (y, t)는 모델간의 데이터 교환을 위한 것이고, (*, t)와 (done, tN)은 시간 동기를 위한 것이다.

### 3.3 DEVS 버스: 이 기종 시뮬레이터의 연동 도구

[8]에서 처음 제안된 DEVS 버스는 DEVS 형식론에 의거한 소프트웨어 버스로 이 기종 시뮬레이터의 연동을 위한 인터페이스를 제공한다 [18]. DEVS 버스는 버스 프로토콜, 버스 제어기, 그리고 버스에 연결된 각 시뮬레이터들로 구성된다 (<그림 7>). 노드 시뮬레이터가 이 기종 시뮬

레이션에 참여하려면, DEVS 버스 프로토콜을 구현하면 된다. DEVS 버스 프로토콜은 DEVS 모델의 실행을 위한 추상화된 시뮬레이션 알고리즘과 유사한 형태로, (x, t), (y, t), (*, t), (done, tN)의 네 가지 메시지로 구성된다. 앞의 두 메시지는 시뮬레이션 노드간의 데이터 전달을 위한 것이고, 뒤의 두 메시지는 버스의 제어를 위한 것이다. DEVS 버스 프로토콜의 기본적인 동작은 DEVS 모델의 추상화된 시뮬레이션 알고리즘과 유사하지만, 분산된 모델을 위해 확장된 것이다.




<그림 7> DEVS 버스의 구조

DEVS 버스 제어기는 Deliverer와 Scheduler로 구성된다. Deliverer는 각 시뮬레이션 노드에서 생성된 출력 사건 (y, t) 메시지를 받아서 대상 노드에 입력 사건으로 (x, t) 메시지를 보내준다. 즉, 시뮬레이션 노드간의 통신이 직접적으로 이루어지는 것이 아니라, Deliverer를 통해 간접적으로 이루어지는 것이다. 이것은, 시뮬레이션 모델의 분할(partition)과 매핑(mapping)에 관계없이 전체 시뮬레이션 모델을 잘 유지하기에 좋은 방법이다. 특히, 이동 에이전트(mobile agent)등의 동적인 구조를 갖는 시스템을 시뮬레이션하기에 유리한 구조이다.

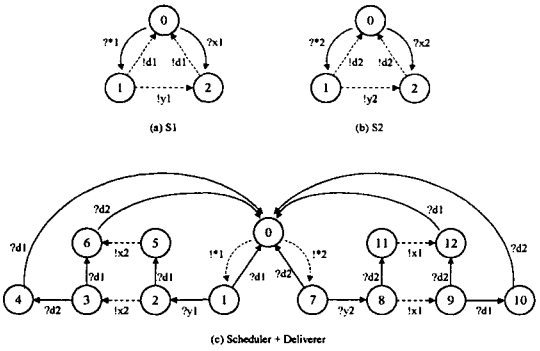
Scheduler는 여러 시뮬레이션 노드들 중에서 다음 번에 버스를 사용할 수 있는 노드를 선정하는 역할을 한다. 이때, 분산 시뮬레이션의 두 가지 동기 방법인 동기식(synchronous) 과 비동기식(asynchronous) 중, 동기식 방법이 사용된다. 각 노드와 Scheduler는 (*, t)와 (done, tN)을 주고받는다. (*, t)는 Scheduler가 특정 노드를 선정해서 버스를 사용하도록 권리를 주는 것이고, (done, tN)은 (*, t)를 받은 노드가 다음 번에 버스를 사용하고자 하는 시간을 예약하기 위한 것이다.

DEVS 버스의 구성 모듈인 노드 시뮬레이터와 버스 제어기의 Scheduler, Deliverer에 대한 시뮬레이션 프로토콜은 다음과 같다.

```
<CS (Component simulator)>
  when receive input event (x, t) from <Deliverer>
    apply external state transition function
    apply time advance function
    report new schedule (done, tN) to <Scheduler>
  when receive output request (*, t) from <Scheduler>
    apply output function and send (y, t) to <Deliverer>
    apply internal state transition function
    apply time advance function
    report new schedule (done, tN) to <Scheduler>
```

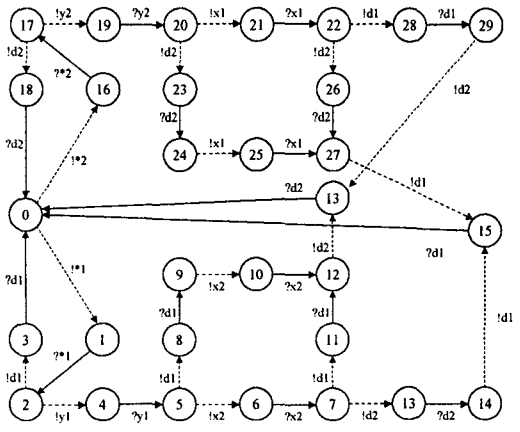
```
<Deliverer>
  when receive an <CS>'s output event (y, t)
    use connection relation and deliver the event as
        input event(s) to connected <CS>
```

```
<Scheduler>
  when receive schedule time (done, tN) from <CS>
    select a <CS> with smallest schedule time
    request an output request (*, t) to the selected <CS>
```




<그림 8> 두 개의 단일 시뮬레이터와 DEVS 버스 제어기의 프로토콜

DEVS 버스 프로토콜의 특성을 설명하기 위해 두 개의 단일 시뮬레이터가 DEVS 버스를 통해 연결되었다고 가정하자. 이때, 각 노드 시뮬레이터의 시뮬레이션 프로토콜이 DEVS 버스 프로토콜과 일치된다고 가정한다. (노드 시뮬레이션 프로토콜과 DEVS 버스 프로토콜사이에 불일치가 있는 경우는 뒤에서 다루기로 한다.) 그렇다면, 각 노드 시뮬레이터의 시뮬레이션 프로토콜을 <그림 8>의 (a), (b)와 같이 나타낼 수 있다. <그림 8>의 (c)는 두 노드 시뮬레이터를 제어하기 위한 버스 제어기로 Scheduler와 Deliverer를 동시에 나타낸 것이다. 그림에서 원은 상태를 나타내고, ?x를 동반한 실선은 외부 사건 x에 의한 상태 천이를 나타내고, !y를 동반한 점선은 내부적으로 상태 천이가 일어나고 외부로 출력 사건 y를 보냄을 나타낸다. *1, x1, y1, d1은 각각 S1의 (*, t), (x, t), (y, t), (done, tN)을 줄여서 나타낸 것이다. 여기서, 노드 시뮬레이터 S1, S2, 그리고 제어기는 비동기적으로 통신하고, 메시지 전송은 신뢰적(reliable)이고 순차적(FIFO)이라고 가정한다. 즉, S1에서는 S2나 제어기의 상태에 상관없이 출력사건을 내보낼 수 있고, 전송된 사건은 손실없이 보낸 순서대로 배달된다. 이때, DEVS 버스는 다음과 같은 특성을 갖는다.



<그림 9> S1, S2, 제어기에 의한 상태 천이도

**Theorem 1.** DEVS 버스 프로토콜을 이용한 시뮬레이션에서는 교착상태(deadlock)가 발생하지 않는다.

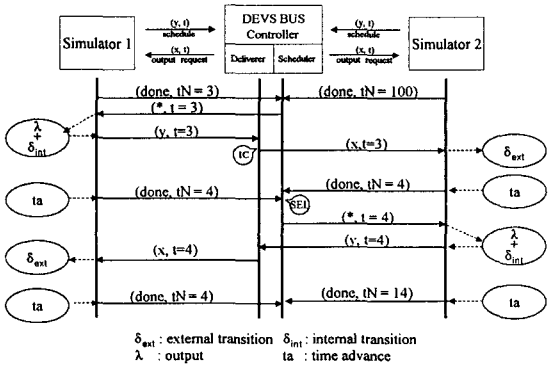
증명: 수학적 귀납법을 이용하여 증명한다.

(i) n = 2 인 경우: 두 노드 시뮬레이터가 DEVS 버스에 연결되었을 때를 고려해보자. <그림 8>의 S1, S2, 제어기를 비동기적으로 결합하였을 때, <그림 9>와 같은 상태 천이도를 얻을 수 있고, 모든 상태에서 다음 상태로의 천이가 존재하므로 교착상태가 발생하지 않음을 알 수 있다.

(ii) n = k 인 경우: k개의 노드 시뮬레이터가 DEVS 버스에 연결되었을 때, 교착상태가 발생하지 않는다고 가정하자.

(iii) n = k+1 인 경우: DEVS 버스 프로토콜을 따르는 시뮬레이션 모델을 DEVS 형식론의 기본모델로 나타낼 수 있다. 이때, 기본모델들은 결합에 대해 닫혀있기 때문에 (closed under coupling) [17], k개의 기본모델을 결합한 결합모델을 한 개의 기본모델로 나타낼 수 있다. 그러므로, k개의 노드 시뮬레이터를 결합하였을 때의 시뮬레이션 프로토콜을 S1, S2와 동일한 형태의 $S_k$로 나타낼 수 있다. 이제 $S_k$, $S_{k+1}$, 그리고 제어기를 결합하면, 역시 <그림 9>와 같은 상태 천이를 하므로, 교착상태가 발생하지 않는다.

(i),(ii),(iii) 으로부터, DEVS 버스 프로토콜을




<그림 10> DEVS 버스 프로토콜을 이용한 시뮬레이션 과정

이용한 시뮬레이션에서는 교착상태가 발생하지 않음을 알 수 있다. Q.E.D.

<그림 10>에는 DEVS 버스 프로토콜을 이용한 시뮬레이션 과정을 나타내었다. 시뮬레이션 처음 단계로, 각 노드들은 (done, tN) 메시지를 Scheduler에게 보낸다. Scheduler는 이중 가장 작은 시간(tN)을 찾아, 현재 시뮬레이션 시각(t)을 tN으로 수정하고, 해당 노드에게 (*, t)를 보낸다. (*, t)를 받은 노드는 자신의 상태를 수정하고 출력 사건 (y, t)를 생성하여 Deliverer로 전달한다. Deliverer는 받은 메시지 (y, t)를 연결 명세 IC를 참조하여 입력 사건 (x, t)로 변환하여 대상 노드로 전달한다. 입력 사건 (x, t)를 받은 노드는 자신의 상태를 수정하고 다음번 시뮬레이션 수행을 위해 (done, tN)을 Scheduler에게 보낸다. 마찬가지로, (*, t)를 받았던 시뮬레이션 노드도 (done, tN)을 Scheduler로 보낸다. Scheduler는 다시 가장 작은 tN을 찾고, 시뮬레이션 시각 t를 tN으로 수정한 후, 해당 노드에 (*, t)를 보낸다. 만약 두 개 이상의 노드에서 동시에 버스 사용을 요청하면, Scheduler는 미리 정해진 우선 순위에 따라 순차적으로 (*, t)를 각 노드에게 보낸다.

### 3.4 시뮬레이션 프로토콜 변환

서로 다른 프로토콜을 사용하는 두 프로토콜 개체가 통신하고자 할 때 프로토콜 불일치가 발생한다. 이것을 해결하는 한 방법으로 프로토콜 변환이 사용된다 [5]. 프로토콜 변환은 두 프로토콜 개체사이에 프로토콜 변환기를 두어 한쪽에서 보낸 메시지를 다른 쪽의 메시지로 변환하여 보내는 방법이다.

이 기종 시뮬레이션에서 각 시뮬레이션 노드들은 서로 다른 시뮬레이션 언어로 기술될 수 있다. 이러한 시뮬레이션 언어를 해석하는 방법을 시뮬레이션 프로토콜이라 한다. 이때, 서로 다른 시뮬레이션 프로토콜을 사용하는 노드들을 상호 연결하기 위해서는 시뮬레이션 프로토콜 변환이 필요하다. DEVS-HLA에서는 DEVS 버스를 통해 시뮬레이션 노드들을 연결하고자 하므로, 기존의 시뮬레이션 프로토콜과 DEVS 버스 프로토콜간의 변환이 필요하다. 본 논문에서는 [11]에서 제안된 프로토콜 변환방법을 사용한다.

먼저, 프로토콜 $P=(P_0, P_1)$와 $Q=(Q_0, Q_1)$가 있다고 가정하자. 이때, $P_0(Q_0)$는 전송자이고, $P_1(Q_1)$은 수신자이다. $P_0(Q_0)$와 $P_1(Q_1)$의 통신은 프로토콜이 일치되지만, $P_0$와 $Q_1$이 서로 통신하고자 할 경우엔 프로토콜 불일치가 발생한다. 이때, 프로토콜 변환기 $C$를 두 개체 사이에 놓아 프로토콜 불일치를 해소시킨다. 이때, 변환기는 다음과 같이 구할 수 있다.

$$C = p(P_0^d \parallel S \parallel Q_1^d)$$

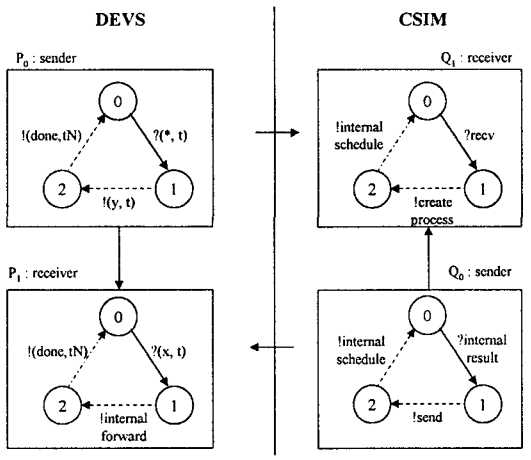
여기서, $P_0^d(Q_1^d)$는 $P_0(Q_1)$의 대칭(dual) 개체로, $P_0(Q_1)$의 입력사건을 출력사건으로, $P_0(Q_1)$의 출력사건을 입력사건으로 갖는다. $S$는 $P_0, C, Q_1$의 통합 시스템이 만족해야 할 서비스의 명세다. 연산 '||' 은 두 개의 FSM (Finite State Machine)을 결합(composition) 하는 연산이고, $p(P_0^d \parallel S \parallel Q_1^d)$는 $C$에 관련된 사건들만을 추출(projection)하는 연산이다. 이렇게 구한 변환기와 두 프로토콜 개체 사이에는 다음과 같은 관계가 성립된다.

**Theorem 2.** 두 프로토콜 개체 $P_0$, $Q_1$와 서비스 명세 $S$가 주어졌을 때, $P_0\|(C\|Q_1)$와 $(P_0\|C)\|Q_1$가 다음과 같은 특성을 갖는 프로토콜 변환기 $C$를 구할 수 있다.

(i) $P_0$와 $(C\|Q_1)$ 사이는 메시지 손실이 없다.

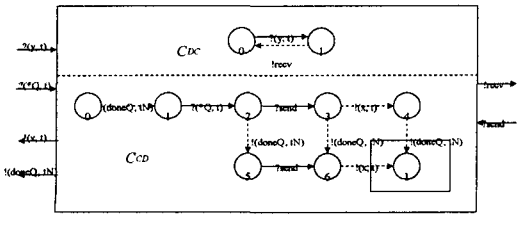
(ii) $(P_0\|C)$와 $Q_1$ 사이는 메시지 손실이 없다.

증명: [11]을 참조.




<그림 11> DEVS, CSIM 시뮬레이션 프로토콜

위 방법을 이용하여, CSIM 시뮬레이션 모델을 DEVS 버스에 연동시킬 수 있는 시뮬레이션 프로토콜 변환기를 구한다. <그림 11>에는 간략화된 DEVS, CSIM 시뮬레이션 프로토콜을 나타내었다. 본 논문에서는 변환기를 두 부분으로 나누어 구한다. 먼저, DEVS 버스에서 CSIM 모델로 메시지를 전송하기 위해선 DEVS 버스의 전송 프로토콜인 $P_0$와 CSIM의 수신 프로토콜인 $Q_1$이 연결되어야 한다. 이때, 변환기 $C_{DC} = p(P_0^d \parallel S \parallel Q_1^d)$가 두 프로토콜 개체사이에 놓여진다. 마찬가지로, $P_1$과 $Q_0$를 연결하는 $C_{CD}$는 $p(Q_0^d \parallel S \parallel P_1^d)$ 으로 구해진다. <그림 12>에는 DEVS 버스에서 CSIM으로의 메시지 전달을 위한 $C_{DC}$와 CSIM에서 DEVS 버스로의 메시지 전달을 위한 $C_{CD}$를 나타내었다.



<그림 12> DEVS/CSIM 프로토콜 변환기

## 4. DEVS-HLA 구현

본장에서는 DEVS 버스를 HLA위에서 구현하는 과정에 대해 기술한다. 현재, DEVS-HLA는 RTI version 1.3.6 [6] 상에서 개발되었다.

### 4.1 DEVS 버스 프로토콜의 구현

DEVS 시뮬레이션 프로토콜을 구현하기 위해 먼저 네 가지 메시지 (x, t), (y, t), (*, t), (done, tN)을 정의해야 한다. HLA에서는 시뮬레이터 사이의 데이터 교환을 위해 객체(object)와 상호작용(interaction)의 두 가지 방법을 제공한다. 본 논문에서 DEVS 버스 프로토콜 메시지는 HLA의 상호작용을 이용하여 구현한다. 네 가지 메시지 상호작용은 다음과 같이 정의한다.

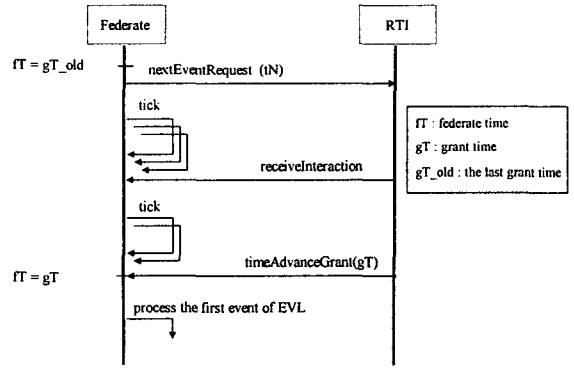
(1) STAR : time, source, destination
(2) X : time, source, destination, port, value
(3) Y : time, source, destination, port, value
(4) DONE : time, source, destination, tN

### 4.2 시뮬레이션 시간 진행 방법

HLA를 이용한 시뮬레이션은 RTI가 제공하는 6가지의 관리 체계를 따르는 것이다. 즉, Federation Management, Declaration Management, Object Management, Ownership Management, Time Management, Data Distribution Management이다. 이중에서, Time Management가 시뮬레이션 시간 진행과 가장 밀접한 관계를 가지고 있다.

HLA의 시간 관리는 규제(time-regulating)와 제약(time-constrained)의 두 가지 요인에 의해 조정된다. 먼저, 규제는 어떤 시뮬레이션 노드의 시뮬레이션 시간이 다른 노드의 시뮬레이션 시간의 진행에 영향을 미침을 나타낸다. 제약은 한 노드의 시뮬레이션 시간이 다른 규약 노드에 영

향을 받음을 의미한다. 본 논문에서는 모든 노드가 규제이면서 동시에 제약인 논리 동기 방법(logically synchronized service)을 사용한다. 이 때, 각 노드들은 DEVS 프로토콜 메시지들을 시뮬레이션 시간 순서대로 정열되도록(timestamp ordered) 서로 주고 받을 수 있다.

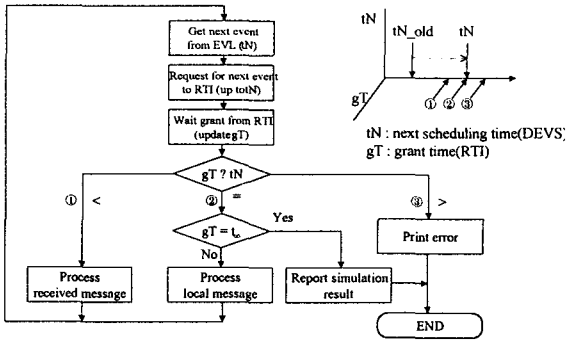



<그림 13> RTI의 시간 진행 방법

<그림 13>에는 RTI의 시뮬레이션 방법으로 사건에 기초한 시간 진행 방법을 나타내었다. fT는 각 노드의 시뮬레이션 시간을 나타내고, tN은 사건 리스트의 가장 처음에 있는 사건의 시뮬레이션 시간이다. 즉, 다음 번에 수행되어야 할 사건의 시뮬레이션 시간이다. gT_old는 마지막으로 RTI로부터 받은 허락 시간(grant time)이고, gT는 최근의 요청에 대한 허락시간이다. 시뮬레이션 진행은 크게 Next Event Request와 Time Advance Grant의 두 가지 함수로 요약된다.

먼저, 각 노드는 다음 번 실행되고자 하는 시간 tN을 결정하고 RTI에 요청(Next Event Request)을 한다. RTI는 이 요청을 받으면, 다른 노드로부터의 사건이 있는지를 확인한다. 만약, 외부로부터의 사건이 있으면 그 시간까지 진행해도 좋다는 허락을 주고, 그렇지 않으면 노드의 요청시간까지 진행해도 좋다는 허락(Time Advance Grant)을 준다.

<그림 14>에는 사건에 기초한 시뮬레이션 알고리즘을 나타내었다. 허락 시간 gT와 요청 시간 tN사이엔 세 가지 경우가 있다. 먼저, gT < tN인 경우는 외부 노드로부터 받은 메시지가 있는 경우이다. gT = tN인 경우는 외부로부터 받은 메시지가 없으므로, 원래 요청했던 사건을 실행할 수 있는 경우이다. 마지막으로, gT > tN인 경우는 시간 동기가 제대로 이루어지지 않았음을 나타낸다.




<그림 14> 제안된 시뮬레이션 알고리즘

### 4.3 사건 정열 (event ordering) 방법

모든 시뮬레이션 사건은 자신이 실행되어야 할 시뮬레이션 시간을 가지고 있다. 만약 두 개 이상의 사건들의 시뮬레이션 시간이 같다면, 이 사건들은 동일한 시뮬레이션 시간에 처리되어야 한다. 단일 노드 시뮬레이션(single node simulation)에서는 이 사건들이 사건의 우선 순위에 따른 실행 순서에 따라 정열이 된다. 그러나, 분산 시뮬레이션에서는 인과 관계만 보전된다면 속도 향상을 위해 여러 개의 시뮬레이션 노드들이 독립적으로 동시 사건들을 실행할 수 있으므로, 이렇게 동시에 실행된 사건들이 새로운 동시 사건들을 발생시킬 수 있다. 그러므로, 분산 시뮬레이션에서는 동시 사건들의 정열 방법이 필요하게 된다.

이 기종 시뮬레이션에서도 여러 노드들이 독립적으로 사건을 수행할 수 있으므로 동시 사건에 대한 정열 방법이 필요하다. 그러나, HLA에서 요구하는 Lookahead 제한과 노드 시뮬레이터에 의한 제약점등으로 기존의 사건 정열 방법들을

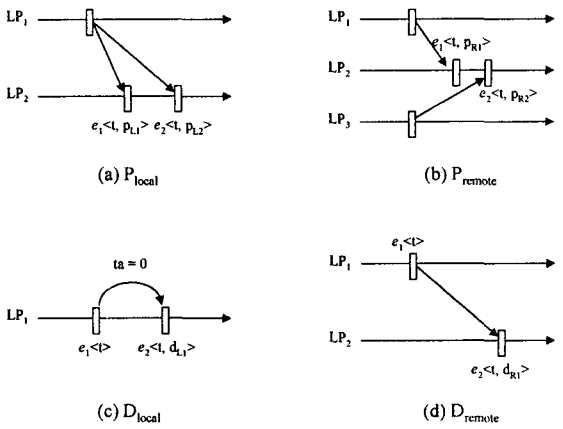
그대로 적용할 수는 없다. Lookahead 제한은 노드 시뮬레이터에서 외부로 보내는 출력 사건의 시간은 노드의 현재 시뮬레이션 시간에 Lookahead 값을 더한 것보다 크거나 같아야 한다는 것이다. 그러나 기존의 노드 시뮬레이터들은 출력 사건의 시간이 노드의 현재 시뮬레이션 시간과 같은 Zero-Lookahead 방식을 사용하므로 HLA의 Lookahead 제한에 위배가 된다. (이것은 HLA의 이전 버전에서 Zero-Lookahead를 지원하지 않았기 때문에 문제가 되었다. 현재 버전의 HLA에서는 Zero-Lookahead를 지원하지만, 성능의 문제가 있을 수 있다. 본 논문에서는 Zero-Lookahead를 지원하지 않는 HLA를 가정한다.)

DEVS-HLA에서 사용하는 사건 정열 방법은 [7]에서 제안된 시간-우선순위 쌍 $tp=\langle t, p\rangle$을 기본으로 다음과 같이 확장된 시간-우선순위 쌍을 사용한다 [12].

$$tp=\langle t, P_{local}, P_{remote}, D_{local}, D_{remote}\rangle$$

이때, t는 시뮬레이션 시간, $P_{local}$ 은 노드내에서의 사건 구별을 위한 우선 순위이고 $P_{remote}$는 원격 노드들로부터의 사건들을 구별하기 위한 우선 순위이다. $D_{local}$은 노드 내에서 발생한 지연(delay)을 해결하기 위한 것이고, $D_{remote}$는 노드간의 메시지 전송으로 생기는 지연을 나타내기 위한 것이다. <그림 15>에는 이 네 가지 속성들을 나타내었다. DEVS-HLA에서는 각각의 속성 값을 다음과 같이 설정한다. 즉, $D_{local}=\varepsilon$, $D_{remote}=L$, $P_{local}=p_L$, $P_{remote}=p_R$이고, $\varepsilon$과 $L$은 시뮬레이션 시간 $t$에 포함시켜 다음과 같은 형태로 사용한다.

$tp=\langle t+\alpha\cdot\varepsilon+\beta\cdot L, p_R, p_L\rangle$. 이때, $\varepsilon$과 $L$은 시뮬레이션의 최소 시간 단위보다 작은 값들이며, $\varepsilon \gg L$의 관계를 만족해야 한다.




<그림 15> 사건정열의 네가지 속성

두 사건 $e_1$과 $e_2$에 대해, 각각의 시간-우선 쌍이 다음과 같다고 하자. 즉, $tp_1=\langle t_1, p_{R1}, p_{L1}\rangle$, $tp_2=\langle t_2, p_{R2}, p_{L2}\rangle$. 이때, 사건 정열은 다음과 같이 이루어진다.

$$tp_1<tp_2 \Leftrightarrow (t_1<t_2 \text{ or } (t_1=t_2 \text{ and } p_{R1}<p_{R2}) \text{ or } (t_1=t_2 \text{ and } p_{R1}=p_{R2} \text{ and } p_{L1}<p_{L2}))$$

어떤 분산 시스템에서 모든 사건이 Lamport의 선행 발생 관계(happened before relation, $\rightarrow$)를 따른다면, 사건 정열이 정확하게 되었다고 한다 [14]. 이때, 다음의 두 가지 시각 조건(clock condition)을 만족하면 선행 발생 관계를 만족한다고 한다.

CC1 : $e_1$, $e_2$가 같은 노드의 사건이고, $e_1\rightarrow e_2$ 이면, $e_1$의 시간(timestamp)이 $e_2$의 시간보다 작아야 한다.

CC2: $e_1$이 한 노드에서 보내는 사건이고 $e_2$가 다른 노드에서 받는 사건이면, $e_1$의 시간이 $e_2$의 시간보다 작아야 한다.

이때, 확장된 시간-우선순위 쌍을 이용한 사건 정열은 다음과 같은 특성을 갖는다.

**Theorem 3.** 제안된 사건 정열 방법은 Lamport의 시각 조건을 만족한다.

증명: (1) CC1의 성립.

(a) 국부(local) 사건의 경우: 한 노드에서의 두 사건 $e_1 = \langle t_1, p_{R1}, p_{L1} \rangle$과 $e_2 = \langle t_2, p_{R2}, p_{L2} \rangle$를 생각해보자. 이때, $e_1 \rightarrow e_2$이라고 하자. 만약, $ta \neq 0$ 이면, $t_2 = t_1 + ta$이고, $ta = 0$이면, $t_2 = t_1 + \varepsilon$ $(\varepsilon > 0)$이므로, $t_2 > t_1$이다.

(b) 원격(remote) 사건의 경우: 한 노드가 RTI로부터 시각 t까지의 시간 진행 허락(Time Advance Grant)을 받았다면, 사건 시간이 t인 모든 사건을 전달받은 것이다. 이때, 이 사건들은 $p_R$과 $p_L$에 의해 유일하게 정열된다.

(2) CC2의 성립.

$e_1$이 한 노드에서 보내는 사건이고 $e_2$가 다른 노드에서 받는 사건이면, 제안된 사건 정열 방법에 의하면, $t_2 = t_1 + L$ $(L > 0)$.

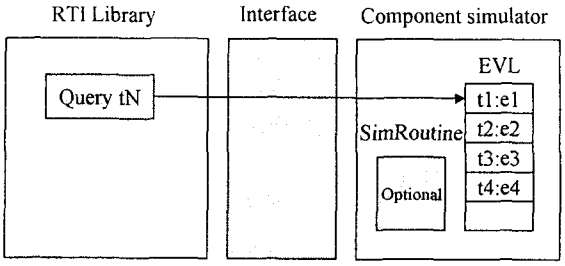
(1), (2)로부터, Lamport 시각 조건이 만족된다. Q.E.D.

**Theorem 4.** 제안된 사건 정열 방법은 동시 사건들을 결정적으로 정열할 수 있다.

증명: 모든 국부사건들은 결정적으로 정열된다. 두 사건 $e_1 = \langle t_1, p_{R1}, p_{L1} \rangle$과 $e_2 = \langle t_2, p_{R2}, p_{L2} \rangle$에 대해, 유일한 비결정적인(non-deterministic) 요인은 지연 도착(delayed arrival)이다. 그러나, 두 사건의 지연 도착여부에 관계없이 이 사건들은 $p_R$과 $p_L$에 의해 유일하게 정열된다. Q.E.D.

### 4.4 단일 시뮬레이션 환경에 대한 요구사항

기존의 단일 시뮬레이션 환경이 DEVS 버스에 연결되기 위해서 크게 세 가지의 조건이 필요하다. 첫째, 외부와의 연결이 가능해야 한다. DEVS-HLA에서는 먼저 DEVS 버스를 통해 다른 노드들과 메시지를 주고받을 수 있어야 한다.




<그림 16> 단일 시뮬레이션 환경에의 요구사항

이를 위해서 단일 시뮬레이션 환경에서 외부로의 경로가 있어야 한다. 가장 간단하게는 외부 라이브러리를 포함할 수 있으면 되고, 화일이나 파이프 형태등도 가능한 방법들이다. 둘째로, 다음 번 실행 예정인 사건의 시뮬레이션 사건을 외부에서 알 수 있어야 한다. 앞에서도 설명했듯이, RTI에서의 시간 진행은 시간 진행 요구에 따른 허락이므로, 각 노드는 다음 시뮬레이션 시간으로 먼저 RTI에게 요청을 해야 한다. 세번째는 시뮬레이션 알고리즘이 바뀌어야 한다. 이것은 필수 사항은 아니지만 좀 더 효율적인 시뮬레이션을 위해선 필요한 것이다. 예를 들어, CSIM의 경우는 시뮬레이션 알고리즘의 수정은 필요하지 않다.

### 4.5 DEVSim++-HLA와 DEVSimJava-HLA

DEVSim++는 DEVS 형식론을 C++로 구현한 것으로 여러 가지 이산 사건 시스템들의 시뮬레이션에 이용되어왔다 [15]. DEVSim++-HLA는 DEVSim++를 HLA에 맞게 확장한 것이다. 시뮬레이션 알고리즘은 RTI의 시간 진행 방법인 요청-허락 방식으로 구현되었고, 외부 입출력 사건에 대한 전달등이 포함되었다.

DEVSimJava는 DEVSim++를 Java 언어로 구현한 것으로 이식성, 상호 운용성, 인터넷 상에서의 실행 능력등을 강화한 것이다 [2]. 특히, 인터넷 상에서 시뮬레이션 모델을 다운로드 하거나 RMI(Remote Method Invocation)를 사용하여 원격지에서 모델을 실행하는 것이 가능하다. DEVSimJava-HLA는 DEVSimJava를 HLA에 맞

도록 확장한 것이다.

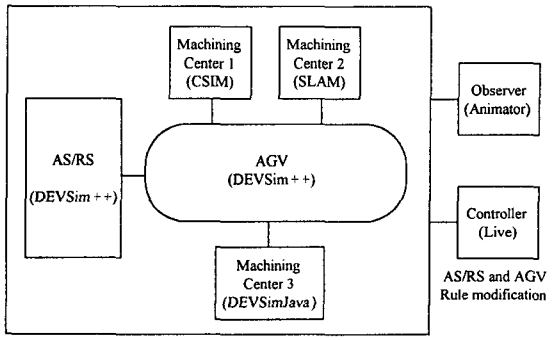
## 5. 적용 사례

### 5.1 유연 생산 시스템

DEVS-HLA의 적용 가능성을 시험하기 위해, 소량 다품종 생산을 목표로 하는 간단한 유연 생산 시스템(Flexible Manufacturing System)에 대한 모델링 및 시뮬레이션을 수행하였다. 대상 시스템은 두 가지 품목을 생산한다. 품목A는 설비 1과 설비2에서 가공이 되고, 품목B는 설비 3에서만 가공된다. 신규 품목은 먼저 자동창고에 저장되고 생산 계획에 따라 가공이 되어 다시 자동창고에 저장되었다가 외부로의 주문이 들어오면 출고된다.

### 5.2 모델링

<그림 17>은 대상 시스템을 모델링한 것이다. 기존에 잘 만들어진 시뮬레이션 모델을 재사용하기 위해 여러 가지 시뮬레이션 환경을 포함한다. 예를 들어, 자동창고와 AGV(Automatic Guided Vehicle)는 DEVSim++ 모델을 사용하고, 설비(Machining Center)는 CSIM, SLAM, DEVSimJava등으로 만들어진 모델을 재사용한다.




<그림 17> 유연 생산 시스템의 모델링

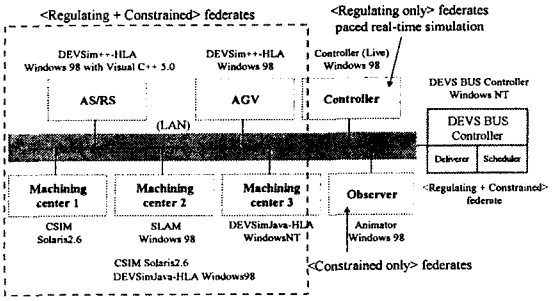
시뮬레이션 도중 운영자가 동적으로 자동창고의 크레인의 움직임이나 AGV의 운용 규칙등 여러 가지 정책들을 변화시킬 수 있도록 하는 제어기, 그리고 전체 시뮬레이션의 진행을 보여주기 위한 Observer도 포함되어 있다. <그림 18>에는 DEVSimJava-HLA에서 실행된 간단한 설비 모델 3을 나타내었다.

```
import DEVSimJava.*;

class MachiningCenter extends AtomicModel{

   initialization ...

   public void extTransfn(StateVars s, double e, Messages message) {
      if ((message.getPort()).equals("in")) {
       Integer part = (Integer)message.getValue();
       if (((Integer)s.getValue("phase")).equals(BUSY)) {
          Error;
       } else {
          s.setValue("part", part);
          s.setValue("phase", BUSY);
       }
     } else
         Error;
   }
   public void intTransfn(StateVars s) {
      if ((Integer)s.getValue("phase")).equals(BUSY)) {
        s.setValue("phase", WAIT);
      } else
         Error;
   }
   public void outputfn(StateVars s, Messages message) {
      if (((Integer)s.getValue("phase")).equals(BUSY)) {
          Integer part = (Integer)s.getValue("part");
          message.setPortVal("done", part);
          message.setPortVal("ready", null);
      } else
         Error;
   }
   public double timeAdvancefn(StateVars s) {
      if (((Integer)s.getValue("phase")).equals(BUSY)) {
         return Rand.exponential_f(ServiceTime, MCSeed);
      } else
          return Model.Infinity;
   }
}
```

<그림 18> DEVSimJava-HLA 설비 모델

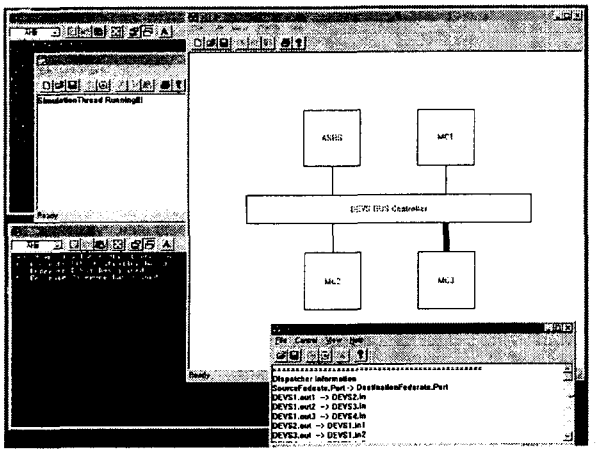
### 5.3 시뮬레이션

이와 같은 복합 모델을 시뮬레이션 하기 위해서 각 노드 시뮬레이터의 시뮬레이션 시간에 대한 고려가 먼저 이루어져야 한다. <그림 19>에 나타냈듯이 자동창고, AGV, 설비, DEVS 버스 제어기등은 규제(time-regulating)인 동시에 제약(time-constrained)이어서 시뮬레이션 사건을 시




<그림 19> 노드 시뮬레이터의 시뮬레이션 시간 진행 방법 구분

뮬레이션 시간 순서(TSO)로 주고받을 수 있다. 운영자에 의해 조정되는 제어기는 실시간으로 다른 노드들에게 영향을 주기 위해 규제가 되고, Observer는 다른 노드들로부터 시뮬레이션 사건들을 수집하기 위해 제약이 된다.



<그림 20> 유연 생산 시스템의 시뮬레이션

<그림 20>에는 대상 시스템의 시뮬레이션 과정을 나타내었다. 우측 상단은 Observer에서 관찰된 전체 시뮬레이션의 간단한 애니메이션 화면이고, 우측 하단은 DEVS 버스 제어기이다. 본 논문의 목표인 상호 운용성은 전체 시뮬레이션이 잘 진행된 것으로 확인되었다. 그러나, 시뮬레이션 수행은 상당한 시간을 필요로 하였다. 이는 현재의 RTI 구현이 최적화 되지 않았고, 전체 시뮬레이션이 동기적 방법으로 진행되기 때문이라 생각된다. 또한, DEVS 버스 제어기로 인한 속도 저하도 한 원인이다. 향후, 개선된 RTI의 사용과 분산된 DEVS 버스 제어기등을 사용하면 시뮬레이션 수행 시간은 좀더 개선될 것으로 생각된다.

## 6. 결론

본 논문에서는 복잡한 대형 시스템을 효과적으로 모델링 및 시뮬레이션 할 수 있는 이 기종 시뮬레이션 틀인 DEVS-HLA에 대해 기술하였다. 가상 소프트웨어 버스인 DEVS 버스를 통해 기존의 단일 시뮬레이션 환경들이 이 기종 시뮬레이션 환경에 연동될 수 있었다. 이때, 시뮬레이션 프로토콜 불일치를 해결하기 위해 프로토콜 변환 기법을 사용하였다. 또한, 이 기종 시뮬레이션을 위한 동시 사건 정열 방법을 제안하여 결정적인 사건 정열을 할 수 있도록 하였다. 제안된 환경의 검증을 위해 간단한 유연 생산 시스템에 대한 모델링 및 시뮬레이션을 수행하였다. 이때, 운영자에 의해 실시간으로 운영 전략을 변화시킬 수 있었다.

이 기종 시뮬레이션은 복잡한 시스템의 해석에 점점 필요한 수단이 되고 있다. 향후, 대규모 시스템의 모델링 및 시뮬레이션에 대한 응용과 다양한 단일 시뮬레이션 환경에 대한 적용이 필요할 것이다.

## 참고문헌


[1] 김용재, 김탁곤, "DEVSim-HLA: DEVS 형식론과 High Level Architecture에 기반을 둔 이 기종 시뮬레이션 환경", 한국시뮬레이션학회 '98 춘계학술대회 논문집, pp. 38-42, 1998

[2] 조정훈, 김탁곤, "DEVSim-Java: Internet /WEB을 기반으로 한 DEVS 모델의 시뮬레이션 환경", 한국시뮬레이션학회 '98 춘계학술대회 논문집, pp. 91-95, 1998

[3] A. Cox et al., "HLA Gateway Status and Future Plans", in *Proceedings of the 1997 Spring Simulation Interoperability Workshop*, 97S-SIW-125

[4] J. S. Dahmann, F. Kuhl, and R. Weatherly, "Standards for Simulation: As Simple As Possible But Not Simple The High Level Architecture For Simulation", *SIMULATION*, 71(6):378-387, 1998

[5] P. E. Green, "Protocol Conversion", *IEEE Tr. on Communications*, COM-34(3) :257-268, 1986

[6] HLA Working Group, IEEE P1516.1/D4, Draft Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification, IEEE, March 1999

[7] Ki Hyung Kim, et al., "Ordering of simultaneous events in distributed DEVS simulation", *Simulation Practice and Theory*, 5:233-268, 1997

[8] Yong Jae Kim and Tag Gon Kim, "A Heterogeneous Distributed Simulation Framework Based on DEVS Formalism", in *Proceedings of the Sixth Annual Conference on AI, Simulation and Planning in High Autonomy Systems AIS'96*, March 23-27, 1996, La Jolla, CA, USA, pp. 116-121

[9] Yong Jae Kim and Tag Gon Kim, "A Heterogeneous Simulation Framework Based on the DEVS BUS and the High Level Architecture", in *Proceedings of the 1998 Winter Simulation Conference*, Dec. 13-16, 1998, Washington,DC, USA, pp. 421-428

[10] Yong Jae Kim, Jeong Hun Cho, and Tag Gon Kim, "DEVS-HLA: Heterogeneous Simulation Framework Using DEVS BUS Implemented on RTI", in *Proceedings of the 1999 Summer Computer Simulation Conference*, July 11-15, 1999, Chicago, IL, USA, pp. 37-42

[11] Yong Jae Kim and Tag Gon Kim, "Circuit-Theoretic View of Protocol Conversion: An Algebraic Approach", Technical Report, SMS Lab, Dept. of EE, KAIST, 1999a

[12] Yong Jae Kim and Tag Gon Kim, "Ordering of Simultaneous Events for Distributed HLA Federates", Technical Report, SMS Lab, Dept. of EE, KAIST, 1999b

[13] Ulrich Klein, et al., "Traffic Simulation Based on the High Level Architecture", in *Proceedings of the 1998 Winter Simulation Conference*, Washington DC, USA, Dec. 13-16, 1998, pp. 1095-1103

[14] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System", *Communications of the ACM*, 21(7):558-565, 1978

[15] Seong Bong Park and Tag Gon Kim, "The DEVS Formalism: Hierarchical Modular Systems Specification in C++", in

*Proceedings of the 1992 European Simulation Multiconference*

[16] C. D. Pham and R. L. Bagrodia, "Building Parallel Time-Constrained HLA Federates: A Case Study with the PARSEC Parallel Simulation Language", in *Proceedings of the 1998 Winter Simulation Conference*, Washington DC, USA, Dec. 13-16, 1998, pp. 1555-1562

[17] B. P. Zeigler, *Multifacetted Modelling and Discrete Event Simulation*, ACADEMIC PRESS, London, 1984

[18] Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim *Theory of Modelling and Simulation* (2nd Ed), ACADEMIC PRESS, San Diego, 2000.

## ● 저자소개 ●

김용재 (e-mail : yjkim@smslab.kaist.ac.kr)
1992년 연세대학교 전기공학과 공학사
1994년 한국과학기술원 전자전산학과 공학석사
1994~현재 한국과학기술원 전자전산학과 박사과정
관심 분야: 모델링/시뮬레이션 방법론, 분산 시스템 해석

김탁곤
1975년 부산대 전자공학과 공학사
1980년 경북대 전자공학과 공학석사
1988년 미국 Univ. of Arizona 전기/컴퓨터공학과 공학박사
1980~1983년 부산수산대학교 (현 부경대학교) 통신공학과 전임강사
1987~1989년 미국 Univ. of Arizona, Environmental Research Lab 연구엔지니어
1989~1991년 미국 Univ. of Kansas 전기.컴퓨터공학과 조교수
1991~현재 한국과학기술원 전자전산학과 조교수/부교수/교수
관심 분야: 모델링 방법론, 이산사건 시스템 시뮬레이션, 컴퓨터/통신 시스템 해석