모델합성 기법을 이용한 시뮬레이션 속도 개선

이 완 복, 김 탁 곤

한국과학기술원 전자전산학과

시스템 모델링 시뮬레이션 연구실

E-mail: wblee@smslab.kaist.ac.kr, tkim@ee.kaist.ac.kr

Composition-based Simulation Speedup Methodology

Wan Bok Lee, Tag Gon Kim

SMSLAB, Dept. of EECS, KAIST

요 약

DEVS 형식론을 비롯한 모듈러한 시스템 모델링 방법은 복잡한 시스템을 모델링 할 때 유리하다. 반면에, 모듈러한 구성요소 모델들은 타 구성요소 모델의 상태 정보를 참조, 복사함으로써 빈번한 메시지 전달을 야기 시켜 시뮬레이션 속도가 저하되는 단점이 있다. 모델 합성법(Composition)은 여러 개의 요소모델들을 하나로 통합시키는 연산으로서 시스템 검증 분야에서 많이 사용되어져 왔다. 본 논문은 모델 합성법을 이용하여 구성요소 모델들 간에 주고받는 메시지 수를 줄이고 시뮬레이션 속도를 개선시키는 방법을 제안한다. 간단한 예제를 통하여 제안한 방법을 자세히 보여주고자 한다.

I. 서 론

컴퓨터 시뮬레이션은 시스템을 구현하기 이전에 시스템의 특성 검증, 성능 평가 등의 목적으로 많이 활용되고 있다. 특히, 시뮬레이션은 계산량이 매우 많으며, 오랜 시간이 걸리기 때문에 예로부터 계산 속도를 개선하기 위하여 많은 방법들이 연구되어 왔다. 병렬 분산 시뮬레이션, 컴파일드 (Compiled) 시뮬레이션 등은 모두 이러한 방법들 중의 하나가 될 수 있다.

본 논문에서는 DEVS 형식론으로 표현되는 시스템 모델들을 빠르게 시뮬레이션 할 수 있는 새로운 방법을 제시한다. 시뮬레이션 과정에는 구성요소 모델들 사이에 많은 이벤트들이 상호 인과 관계에 의하여 통신하게 된다. 본 논문에서는 연관성이 많은 구성요소 모델들을 하나의 모델로 합성함으로써 통신하는 메시지 수를 줄이며 전체 시뮬레이션 속도를 개선하게 되었다. 이를 위해 DEVS 구성요소 모델들 간의 통신의미 (Communication Semantics)를 명확히 정하고, 모델합성 기법을 정의하였다.

특히, DEVS 모델에서는 시스템이 모듈러하게 모델링 되기 때문에 타 개체의 변수 값을 참조하기 위하여 상호 많은 이벤트들을 주고 받게 된다. 이러한 빈번한 이벤트 통신은 시뮬레이션 속도를 저하시킬 수 있는데, 본 논문에서 제안하는 방법은 이러한 단점을 극복할 수 있게 해 준다.

본 논문은 다음과 같이 구성되었다. 2장에서는 여

기에서 논의하는 이산사건 시스템 모델과 시뮬레이션 알고리즘을 소개하고, 3장에서는 모델합성 기법에 대해서 정의한다. 4장에서는 모델합성 기법을 적용하였을 시 시뮬레이션 속도가 개선될 수 있음을 간단한 예제실험을 통하여 보이고, 5장에서 결론을 맺는다.

II. DEVS 형식론과 시뮬레이션

2.1 DEVS 형식론

DEVS 형식론은 이산사건 시스템을 모듈러하고도 계층적으로 모델링 할 수 있는 방법을 제시한다. 이 형식론에는 모델을 계층적으로 분해할 수 있도록 원자 모델(Atomic Model)과 결합 모델(Coupled Model)의 두 가지 클래스로 나누어서 모델을 표현하고 있다.

원자 모델은 더 이상 나눌 수 없는 모델로서 각 구성요소의 동적 특성을 시간명세 상태전이시스템(timed state transition system) 레벨에서 기술한다. 원자 모델의 수학적 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 = $\{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$
: total state of AM(e: elapsed time)

결합 모델은 구성요소 모델들 간의 연결 관계를 표현하는 모델로서 구성요소 모델들을 조립하여 더 큰 단위의 모델을 만들 수 있게 한다.

$CM = \langle X, Y, \{M_i\}, EIC, EOC, IC, SELECT \rangle$
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
$SELECT$: $2^{(M_i)} - \phi \rightarrow \{M_i\}$

DEVS 형식론에 대한 자세한 내용은 [2][3]에서 참조할 수 있다.

2.2 추상화된 시뮬레이터

DEVS 모델을 실행하는 추상화된 시뮬레이터 알고리즘은 Zeigler에 의해 제안되었다[2]. 이 알고리즘에는 원자 모델을 실행하는 *Simulator*, 결합 모델을 실행하는 *Coordinator*라고 하는 프로세스가 있어서 정해진 알고리즘에 의해 해당 모델의 특성함수들을 실행하여 시뮬레이션을 수행한다.

*Simulator*의 알고리즘은 다음과 같다.

```
When receive an input (x,t)
    done: = false
    if   t_L ≤ t ≤ t_N  then
         e: = t − t_L
         s: = δ_ext(s, e, x)
         t_L: = t
         t_N: = t_L + ta(s)
    else error
    done: = true
end when

When receive an input (*,t)
    done: = false
    if   t = t_N  then
         Y: = λ(s)
         s: = δ_int(s)
         t_L: = t
         t_N: = t_L + ta(s)
    else error
    done: = true
end when
```

*Coordinator*의 알고리즘은 다음과 같다.

```
When receive an input (x,t)
    done: = false
    if  t_L ≤ t ≤ t_N  then
        send input (x,t) to each component
        simulator i;
        wait until all simulators done;
        t_L: = t
        t_N: = minimum of component t_N s
    else error
    done: = true
end when

When receive an input (*,t)
    done: = false
    if  t = t_N  then
        find the simulators with minimum
          t_N ;
        SELECT one, i* , and the input
        (*,t) to it;
        send the signals (x_{i*,j}, t) to each
        of its influencees j;
        wait until this simulator i* and
        each of its influencees done;
        t_L: = t
        t_N: = minimum of component t_N s
    else error
    done: = true
end when
```

이 추상화 시뮬레이터 알고리즘을 C++로 구현한 것이 devsim++ 시뮬레이션 환경이다[3]. Devsim++ 환경에서는 Int_Event Ext_Event, Done_MSG 라는 세 종류의 메시지 전달을 통하여 시뮬레이션이 진행된다. Int_Event는 어떤 개체가 내부 상태 전이를 할 시간이 되었을 경우 시뮬레이션 엔진 내부의 스케줄러가 발생시키는 메시지이며, Ext_Event는 어떤 개체의 출력 이벤트가 다른 개체들의 입력 이벤트와 연결 관계가 있을 경우에 발생하는 메시지이다. Done_Msg는 다음 스케줄 시간을 정하기 위해서 하위 계층의 시뮬레이터가 상위 계층의 시뮬레이터에게 보내주는 메시지이다.

시뮬레이션 과정에서는 위 세 메세지가 상호 인과관계에 의해서 연속적으로 발생하게 된다. 그러므로 구성관계가 복잡하고 상호 작용(interaction)이 많은 모델들 간에는 많은 메시지들이 시뮬레이션 도중에 발생하며, 그 만큼 많은 시간이 소요되게 된다.

다음 절에서 설명하는 모델합성 기법은 모델들 간의 상호작용을 상당히 줄이고 이러한 통신 메시지들을 줄일 수 있는 방법이다.

Ⅲ. DEVS 모델 합성법

모델 합성법(Composition)은 시스템 검증(Verification) 분야에서도 많이 이용되어 왔다. 대부분의 경우 해석 모델의 정적해석(Static Analysis)을 위해 전체 시스템의 상태도달 그래프(Reachability Graph)를 생성하는데 이용되었다.

본 논문에서 제안하는 모델 합성법은 정적해석에서 사용되는 모델 합성법과 차이점이 있다. 시뮬레이션 모델은 해석모델과 달리 각 모델에서 사용하는 상태 변수의 값을 실행시간(Run Time)에 계산하면 충분한, 동적 모델(Dynamic Model)이기 때문에 정적해석에서처럼 모든 발생 가능한 값들을 계산하여 상태도달 그래프를 구해낼 필요가 없다.

3.1 통신의미 (Communication Semantics)

모델합성을 정의하려면, 먼저 전제되는 통신의미(Communication Semantics)를 분명히 해야 한다[1]. 전통적인 시스템 검증 방법론 분야에서 사용되어 지는 Automata, Process Algebra 등의 모델들은 공유이벤트(shared event)로써 통신의미를 규정하고 있다. 공유이벤트는 모든 구성요소에서 동시에 발생할 수 있는 경우에 한해서만 발생할 수 있다. 그러므로 어떤 하나의 구성요소 모델이 현재 상태에서 공유이벤트를 수용(Accept)할 수 없다면 그 공유이벤트는 발생할 수 없으며, 경우에 따라서

는 교착상태(dead lock)가 발생하기도 한다.

공유이벤트를 이용한 통신의미는 일반적으로 IO 시스템의 통신의미에 부합되지 않는다[5]. 예를 들어, 두 개의 구성요소 모델에서 같은 라벨(label)의 출력 이벤트를 가지고 있을 경우, 이들이 발생했다면 동기화된 것을 의미하는데, 이것은 물리 세계에서 수용되기 어려운 현상이다.

이러한 문제점 때문에 Lynch는 IO 시스템 모델의 일종인, IO Automata를 제안할 때 기존 공유이벤트를 이용하여 시스템을 모델링하되, Compatability 라는 별도의 조건이 만족되도록 요구하고 있다[4]. Compatible한 IO Automata 에서는 모든 내부 이벤트(internal event)들이 서로 다른 라벨(label)을 가져야 하는 등 여러 제한 조건을 갖지만, IO 시스템의 일반적인 특성을 모델링 할 수 있다.

저자가 알고 있는 바로는, 현재까지 DEVS 모델의 통신 의미에 대해서 논의된 바가 없다. DEVS 모델은 IO 시스템 모델이기 때문에, IO 시스템의 의미와 부합하는 통신의미가 필요하다. 이러한 필요성 때문에 저자는 약한 동기화 라는 통신 의미를 제안한다. 약한 동기화는 Heymann이[1] 제안한 Prioritized Synchronization의 의미와 일맥 상통하는 바가 많다. IO Automata에서는 Compatible한 모델들만이 유효한 의미를 갖는 반면, 약한 동기화의 가정에서는 모든 DEVS 모델들이 유효한 의미를 가진다.

정의1) 약한 동기화 (Weak Synchronization)

시스템을 구성하는 이벤트가 세 종류의 형태, 입력, 출력, 내부 이벤트로 나뉘어 지며 다음의 특성들을 만족할 때 시스템의 통신 의미는 약한 동기화라 한다.

1) 출력 이벤트와 내부 이벤트는 블록되지 않고 구성요소 자체에서 언제든지 발생시킬 수 있다. (spontaneous output)

2) 입력 이벤트는 항상 같은 라벨을 가지는 출력 이벤트와 동기화 되어서 발생되며 외부 환경에 의해서만 발생될 수 있다. (passive input).

본 논문에서 다루는 모델 합성법은 약한 동기화 가정 하에서 정의 되었다.

3.2 모델합성(Composition)

하나의 결합 모델을 구성하는 N 개의 DEVS 원자 모델과 결합 모델에 명세되어 있는 구성요소들의 연결관계, SELECT 함수가 주어지면, 하나의 원자 모델로 합쳐질 수 있는데 이 과정을 모델합성이라 한다. 모델합성 과정에서 몇몇 이벤트들은 내부화(internalization) 과정을 거칠 수 있다. 그림 1 에서 Buff의 "bout" 출력 이벤트는 결합모델 내부에서만 연결 관계가 있기 때문에 내부화되어진다.

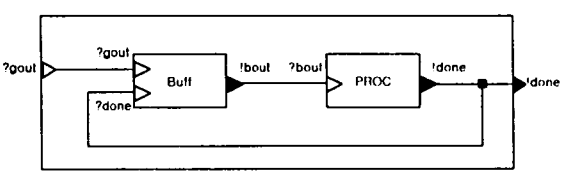


그림 1 결합 모델 예제

모델 합성과정에는, 구성 원자 모델들의 명세, 구성 모델들 간의 연결 관계, 내부화되는 입출력 이벤트 집합, 그리고 SELECT 함수의 명세가 필요하다. 본 논문에서는 연결 관계 정보를 간단히 하기 위해 같은 라벨을 가지는 이벤트들은 연결 관계가 있다고 가정한다. 내부화되는 입출력 이벤트 집합과 SELECT함수는 결합 모델로부터 얻어진다.

정의2) 모델합성 (Composition)

$M_1 = \langle X_1, S_1, Y_1, \delta^1_{int}, \delta^1_{ext}, \lambda_1, ta_1 \rangle, \ldots$

$M_n = \langle X_n, S_n, Y_n, \delta^n_{int}, \delta^n_{ext}, \lambda_n, ta_n \rangle$은 DEVS 원자 모델들이며, 내부화되는 입력이벤트 집합, 출력이벤트집합을 IX, IY라고 하고, 다음 스케줄 시간에 선택되어지는 구성요소 모델의 인덱스를

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

$X\subseteq X_1\bigcup\ldots\bigcup X_n$
$Y\subseteq Y_1\bigcup\ldots\bigcup Y_n$
$S\subseteq S_1\times R\times,\ldots,S_n\times R$
$s=(s_1, e_1, \ldots, s_n, e_n)$
$\lambda(s_1, e_1, \ldots, s_n, e_n)=\lambda_i(s_i)\wedge i=nsmi\wedge\lambda_i(s_i)\notin IY$
$ta(s_1, e_1, \ldots, s_n, e_n)=Min_i(ta_i(s_i)-e_i)$

δ_{ext} is defined as
$\delta_{ext}((s_1, e_1, \ldots, s_n, e_n), e, a)=(s_1', e_1', \ldots, s_n', e_n')$
여기서 s_i', e_i'는 다음과 같다.
경우1) $a\in X_i$인 경우 $s_i'=(\delta_{ext}^i)'((s_i, e_i), e, a); e_i'=0$
경우2) $a\notin X_i$인 경우 $s_i'=s_i; e_i=e_i+e$

δ_{int} is defined as
$\delta_{int}((s_1, e_1, \ldots, s_n, e_n), e, a)=(s_1', e_1', \ldots, s_n', e_n')$
여기서 s_i', e_i'는 다음과 같다.
경우1) $nsmi=i$ 인 경우 $s_i'=\delta_{int}^i(s_i, e_i); e_i'=0$
경우2) $i\in$ influencees$(j, a)\wedge j=nsmi$ 인 경우 $s_i'=\delta_{ext}^i((s_i, e_i), e, a); e_i'=0$
경우3) 그 밖의 경우 $s_i'=s_i; e_i'=e_i+ta_j(s_j, e_j)$ where $j=nsmi$

nsmi(Next Scheduled Model Index의 약자임) 라고 하자. 합성모델은 약한 동기화의 가정에서 다음과 같다. 합성된 모델의 상태집합은 단순히 각 구성요소 모델들의 상태집합의 조합(Cartesian Product)이 아니라, 각 구성요소 모델의 elapsed time이 별도로 들어간다. 이것은 각 이벤트들의 다음 스케줄 시간이 elapsed time에 의존할 수 있기 때문인데, 다른 말로 timing dependancy 라고 말한다. 위에서 *influencees(j, α)*는 *j*번째 구성요소 모델에서 발생하는 이벤트 α 가 영향을 주는 구성요소들의 인덱스 집합이다.

*nsmi*는 다음 스케줄 시간을 가지고 있는 구성요소 모델의 인덱스 값이다. 이 값은, 합성모델을 구성하는 원자 모델들의 다음 스케줄 값중 가장 작은 것을 가진 모델의 인덱스 값이다. 만일, 가장 작은 값이 하나 이상이라면, SELECT 함수에 의해 하나로 선택이 된다. 모델 합성의 결과로 얻어지는 새로운 원자 모델은 항상 deterministic 하다. 이것은 한 시점에서 여러 개의 구성요소 모델에서 internal transition들이 발생하는 상황이라 하더라도 결합 모델의 SELECT 함수에 의해 오직 하나의 transition 만이 선택되는 것을 *nsmi*가 반영하기 때문이다.

합성된 모델은 일반적으로 주어진 구성요소 모델들보다 복잡해지기 때문에 디버깅이 어려워질 수 있다. 반면에, 합성모델은 메시지 통신이 적고 내부화(Internalization) 과정에 의해 특성 함수들이 결합하기 때문에 그만큼 빨리 실행될 수 있다. 다음 장은 모델합성 과정을 통해 시뮬레이션 속도가 개선됨을 실험을 통해 보인다.

Ⅳ. 모델합성 기법을 이용한 시뮬레이션 속도 개선

합성된 모델이 더 빨리 수행될 수 있음을 보이기 위하여 간단한 Buf-Proc 예제에 대해 실험을 해 보았다. Buf-Proc 모델에 대한 자세한 설명은 [3]을 참조할 수 있다.

4.1 실험 설계

Buf-Proc 시스템을 아래 그림 2에서 보는 바와 같이 네 가지 경우로 구성하였다. 구성1은 모델합성

과정이 적용되지 않은 일반적인 구성이며, 구성2는 Buff와 Proc 모델이 합성된 BP 모델이 Buff와 Proc 모델을 치환한 것이며, 구성3은 BP와 Genr이 결합된 GBP가 사용된 모델이다. 마지막으로 구성4는 네 개의 구성요소 모델이 하나로 합성된 GBPT가 사용된 것이다. 실험에 사용한 도구는 KAIST SMSLAB에서 개발된 Devsim++ 시뮬레이션 환경이다[3].

실험에 사용한 파라메터는 다음과 같다. "gout" 이벤트의 발생 간격은 λ=100 인 exponential 분포로, Proc의 서비스 시간은 λ=75 인 exponential 분포로 정하였다. 1000개의 서비스가 Proc에서 이루어지면 시뮬레이션은 종료된다.

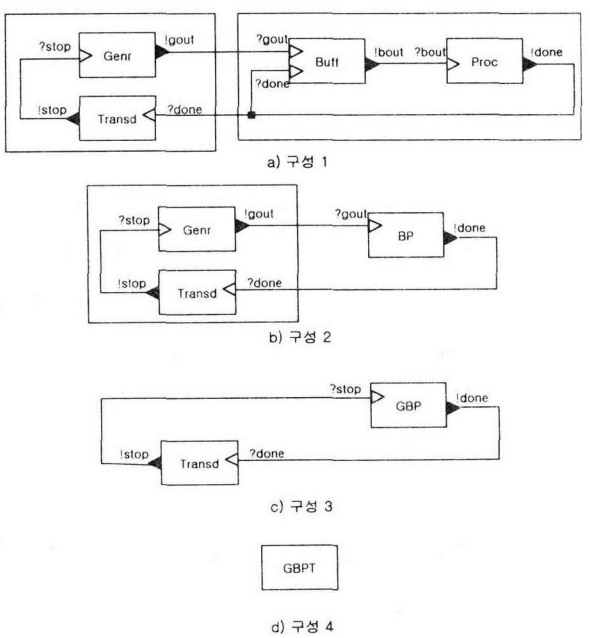
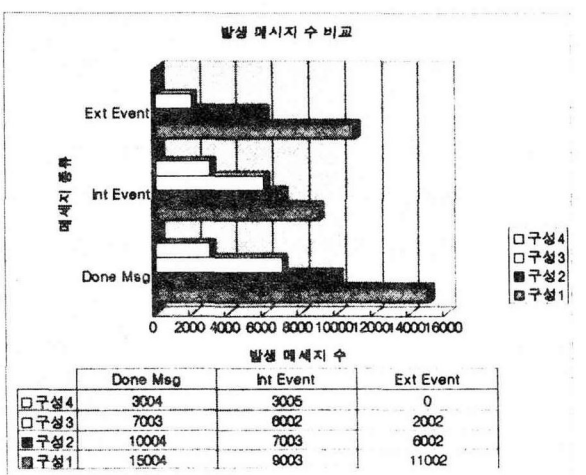


그림 2 모델 구성의 여러 경우들

4.2 결과 분석

모델합성 과정이 시뮬레이션 속도에 미치는 영향을 파악하기 위해, 4.1절의 각 경우에 대해 시뮬레이션 도중 발생하는 메시지 수와 전체 실행 시간을 비교해 보았다. 실험 결과의 신뢰도를 높이기 위해 여러 번 실험한 후 측정치의 평균을 취하는 것이 바람직하겠으나, 여기서는 간단히 전체 시뮬레이션 시간을 충분히 길게 하여 측정치를 수집하였다.

그림 3과 그림 4의 실험 결과를 살펴보면 모델합성이 많이 적용될수록 빨리 시뮬레이션 됨을 알 수 있다. 모델합성이 많을수록 발생하는 메시지 수가 줄어들었으며, 시뮬레이션 시간도 더욱 단축되었다. 시뮬레이션 시간이 메시지 수가 줄어든 비율만큼 단축되지는 않았는데, 이것은 전체 시뮬레이션 과정이 단순히 메시지 라우팅만으로 이루어지는 것이 아니라 과 각 모델들의 특성함수 실행에도 시간이 소요되기 때문인 것으로 분석된다.



	Done Msg	Int Event	Ext Event
□구성4	3004	3005	0
□구성3	7003	6002	2002
■구성2	10004	7003	6002
▨구성1	15004	9003	11002

그림 3 시뮬레이션 과정에서 발생하는 메시지 수 비교

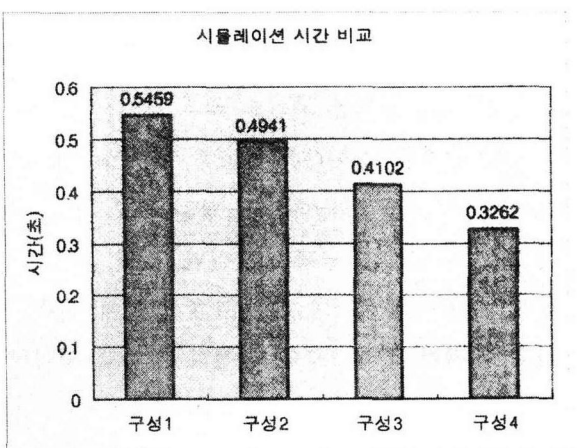


그림 4 시뮬레이션 소요 시간 비교

V. 결 론

모듈러 시스템 모델링은 복잡한 시스템을 체계적으로 모델링 하는데 유리하나 타 개체의 상태 변수를 참조하기 위하여 많은 이벤트를 주고 받게 된다. 이로 인해 시뮬레이션 속도가 저하될 수 있는 단점이 있다. 본 논문에서 제안한 모델 합성법은 구성 요소모델들 간에 통신하는 메시지 수를 줄이고 이벤트 내부화 과정을 통하여 시뮬레이션 속도 개선을 가능케 한다. 또한 제안한 방법은 자동화된 틀로 구현할 수 있다.

참 고 문 헌

[1] Michael Heymann, Concurrency and Discrete Event Control, *IEEE Control Systems Magazine*, Vol. 10, No. 4, pp. 103-112, 1990.

[2] B. P Zeigler, *Multifacetted Modeling and Discrete Event Simulation*, Orlando, FL, Academic PRess, 1984

[3] Tag Gon Kim, *Devsim++ Users Manual*, ftp://smslab.kaist.ac.kr/pub/DEVSim++-1.0/PC/DEVSim.zip

[4] N. A. Lynch and M. R. Tuttle. An introduction to Input/Output automata. *CWI-Quarterly*, 2(3):219--246, Sept. 1989.

[5] Frits W. Vaandrager. On the Relationship Between Process Algebra and Input/Output Automata. *LICS* 1991: 387-398