한국시뮬레이션학회 논문지
Vol. 20, No. 1, pp. 87-96 (2011. 3)

이벤트 지향 DEVS 실행 환경의 설계, 구현 및 성능 비교

권세중[1†] · 김탁곤[1]

Design, Implementation and Performance Analysis of Event-oriented Execution Environment for DEVS.

Se Jung Kwon · Tag Gon Kim

ABSTRACT

DEVS(Discrete Event Systems Specification) is a set theoretic formalism developed for specifying discrete event system. For execution of DEVS, we need an execution environment, which consists of simulation engine and models interpreted by the simulation engine. Common existing environments use hierarchical scheduling algorithm for DEVS execution. This hierarchical scheduling is a proper algorithm for DEVS execution because of hierarchical and modular characteristics. But this algorithm has overheads owing to message passing and time management. To overcome these overheads, we apply event-oriented simulation to DEVS execution and we remove hierarchical overheads. In event-oriented simulation, the scheduling of model execution is performed by events and event list. We propose three event-oriented execution environments for DEVS and experiment about the performance of our proposed environments in comparison with the existing execution environment using the hierarchical scheduling. The experimental results show our environments works better than existing environment using the hierarchical scheduling.

Key words : DEVS Formalism, Discrete Event System, Simulation Speedup, Event-oriented Simulation

요 약

DEVS는 이산 사건 시스템을 명세하는 집합론에 근거한 수학적인 틀이다. 이러한 DEVS를 통해 시스템을 시뮬레이션할 때는 명세된 DEVS로부터 구현된 모델과 시뮬레이션 엔진으로 이루어진 실행 환경이 필요하다. 시뮬레이션 엔진은 스케줄링 알고리즘을 통해 모델을 해석하고 실행하는데, 주로 DEVS의 계층적이고 모듈러한 특성을 따라 계층적인 스케줄링 방법을 사용한다. 그러나 이 방법은 계층적인 구조로 인해 시간 관리와 메시지 패싱에 있어서 오버헤드가 존재한다. 본 논문에서는 이러한 오버헤드를 해결하기 위해 이벤트 지향 시뮬레이션을 적용한다. 이벤트 지향의 시뮬레이션은 이벤트에 의해서 모델이 실행되며 이벤트 리스트에 의해 이벤트가 관리된다. 이벤트 리스트에 의해서 시간 관리가 되고 메시지 패싱이 이루어지기 때문에 계층적인 오버헤드를 줄이고 성능 향상을 이룰 수 있다. 본 논문은 이벤트 리스트를 이용한 실행 환경을 제안하고 이러한 실행 환경을 위한 모델 구현 방안에 대해 논하며, 각 실행 환경의 성능을 비교, 분석한다.

주요어 : DEVS 형식론, 이산사건시스템, 시뮬레이션 성능향상, 이벤트 지향 시뮬레이션

1. 서 론

이산 사건 시스템이란 시스템 내부 혹은 외부 요인으로 인하여 불규칙적으로 실수 시간에 일어나는 이벤트에 의해서 상태가 변화하는 시스템이다. 이러한 이산 사건 시스템은 통신 모델, 물류 시스템, 생산 시스템, 교통 시스템, 워 게임 등에서 찾아 볼 수 있다.

이러한 이산 사건 시스템의 행동이나 성능을 분석하기 위해서는 모델링을 통해 모델을 만든다. 모델링을 위해서 수학적인 틀(Framework)을 활용하는 방법이 주로 많이 쓰이는데 대표적인 이산 사건 시스템의 수학적인 공식에는 이산사건 시스템 명세(Discrete Event System Specification,

*본 연구는 문화체육관광부 및 한국콘텐츠진흥원의 2009년도 문화콘텐츠산업기술지원사업의 연구결과로 수행되었음.
접수일(2010년 12월 18일), 심사일(1차 : 2011년 3월 8일), 게재 확정일(2011년 3월 28일)
[1] KAIST 전기및전자공학과
주 저 자 : 권세중
교신저자 : 권세중
E-mail; sjkwon@smslab.kaist.ac.kr

DEVS)가 있다. DEVS는 3개의 집합과 4개의 함수로 수학적인 틀을 제공하며 시스템을 모듈화하고 계층적으로 결합하여 표현한다[1].

DEVS를 실행하여 시뮬레이션 결과를 얻기 위해서는 시뮬레이션 알고리즘과 그에 맞게 DEVS로부터 구현된 모델 구현물이 필요하며 이 두 가지를 합쳐서 실행 환경이라고 한다.

일반적으로 DEVS는 계층적이고 모듈러한 구조를 가지고 있기 때문에 DEVS 구현물 또한 같은 구조를 가지는 것이 구현하기에 편하다. 따라서 이런 모델을 실행하기 위해서 알고리즘 또한 비슷한 구조로 계층적으로 스케줄하는 것이 주로 쓰인다.

계층적 스케줄링 알고리즘은 계층적이고 모듈러한 DEVS의 구조를 적용하여 시뮬레이터를 구현한다. 이러한 알고리즘은 DEVS의 원자 모델에 대응되는 시뮬레이터(Simulator)와 결합모델에 대응되는 코디네이터(Coordinator)가 존재하여 구현된 DEVS 모델의 함수를 호출하고 결과를 돌려받으면서 시뮬레이션을 진행한다. 이때 명세된 DEVS로부터 구현된 모델은 수동적인 형태로 오직 시뮬레이션 엔진의 호출을 받고 결과를 돌려준다.

시뮬레이션 알고리즘은 모델을 정해진 시간에 실행시키는 시간 관리와 모델의 출력을 다른 모델의 입력으로 연결시켜주는 메시지 패싱(Message Passing)을 해야 하는데, 계층적인 스케줄링은 DEVS와 실행 환경이 같은 구조를 가진다는 장점이 있지만 계층적인 구조로 인한 오버헤드가 발생한다.

계층적인 스케줄링에서 시간 관리는 하위 모델로부터 상위 모델로 정보가 모아지는 형태로 일어나고 메시지 패싱은 계층적인 구조를 따라 상위 모델을 거쳐서 전달되는 구조를 가진다. 따라서 계층적 구조가 깊어지고 복잡해질수록 중간 과정이 늘어나게 되고 성능이 느려지게 된다.

이러한 오버헤드를 해결하기 위해 모델의 구조를 바꾸거나 모델을 합성하는 방법들이 제안되었다[2,3]. 이러한 방법들을 통해 상당부분 성능 향상을 가져올 수 있으나 내부적으로는 계층적 알고리즘을 사용함으로써 계층적 알고리즘이 지니는 한계를 동일하게 가진다.

본 논문에서는 위와는 다르게 실행 환경의 시뮬레이션 알고리즘에 이벤트 지향 시뮬레이션을 적용해서 DEVS 실행의 성능 향상을 가능하게 하는 이벤트 리스트를 사용한 실행 환경을 제시한다.

이벤트 지향 시뮬레이션에서 스케줄링은 이벤트 리스트에 의해서 이루어지고 모델의 실행은 이벤트에 의해서 실행된다. 이벤트는 시간과 실행 대상이 되는 모델 정보를 가지고 있다. 이러한 이벤트는 이벤트 리스트에 시간 순으로 정렬되어 존재하며 각 이벤트에 의해서 모델이 실행되고 모델은 다음 실행할 모델의 이벤트를 발생시킨다. 시뮬레이션 엔진은 이 작업을 반복하면서 모델의 실행을 계속한다[4]. 이러한 이벤트 리스트를 사용한 실행 환경에서 DEVS 명세로부터 구현된 모델은 계층적인 구조가 필요없고 따라서 앞서 언급했던 계층적인 오버헤드를 제거할 수 있게 된다.

하지만 성능 향상이 있는 대신에 사용자가 모델을 구현하기에 비교적 불편하다는 문제와 DEVS의 모듈러한 특성을 잃어버리는 단점이 있다. 따라서 이벤트 지향의 모델을 구현하는 것과 함께 객체 지향, 기존의 계층적인 모델을 구현하여 이러한 단점을 해결하면서도 여전히 계층적인 오버헤드를 제거하는 방법 또한 제안한다.

본 논문에서는 이벤트 리스트를 사용한 실행 환경을 제안하고 계층적인 스케줄링의 실행 환경에 비해 성능 향상이 있다는 것을 보인다. 또한 이벤트 리스트를 사용한 실행 환경을 위한 여러 가지 모델 구현 방법을 살펴보고 장단점을 분석하고 성능을 분석한다.

본 논문의 순서는 다음과 같다. 2장에서 DEVS 형식론과 계층적인 알고리즘에 대한 배경 지식을 설명하고 3장에서는 이벤트 리스트를 이용한 스케줄링 알고리즘을 설명한다. 4장에서는 이러한 실행 환경에 쓰이는 DEVS 모델에 대해 설명하고 문제점을 논하고 5장에서는 이 문제에 대한 절충안을 제시한다. 6장에서는 사례 연구를 통해 성능 향상의 정도를 보여주고 마지막으로 7장에서는 결론을 맺는다.

2. DEVS 형식론

DEVS 형식론은 이산 사건 시스템을 객체 단위로 모듈화하여 계층적으로 결합하여 표현하는 집합론에 근거한 수학적인 틀이다. DEVS 형식론에는 시스템의 기본적인 구성 요소를 나타내는 원자 모델(Atomic Model)과 결합 모델(Coupled Model)이 있다[1].

2.1 원자 모델

원자 모델은 계층적인 구조에서 가장 하위의 기본적인 모듈로서 시스템의 행동을 기술한다. 3개의 집합과 4개의 함수로 이루어져 있으며 원자 모델 M의 수학적인 표현은 다음과 같다.

$M = <X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta>$
X : 이산사건 입력 집합
Y : 이산사건 출력 집합
S : 일련의 이산사건 상태의 집합
δ_{ext} : $Q \times X \rightarrow S$: 외부 상태 천이 함수
$Q = \{(s,e) \mid s \in S, 0 \leq e \leq ta(s)\}$: total state of M
δ_{int} : $Q \rightarrow Q$: 내부 상태 천이 함수
λ : $Q \rightarrow Y$: 출력 함수
ta : $S \rightarrow R_{0,\infty}^{+}$: 시간 진행 함수

Atomic 모델은 대상 시스템의 상태를 집합 S로 표현한다. 시스템 이론에서 이 상태의 변화는 두 가지 경우로 나눌 수 있는데 입력이 있는 경우와 없는 경우로 나눌 수 있는데, 그럴 때, Atomic 모델의 4가지 함수의 호출 순서는 다음과 같다.

먼저 정해진 시간 동안 입력이 없을 경우에, λ함수가 실행되어 출력 Y를 내보내게 된다. 그리고 내부 상태 천이 함수 δ_{int}를 실행 시켜 상태를 바꾸고 ta함수를 실행시켜 다음 출력이 발생할 때까지의 시간 간격을 정해준다.

출력 Y는 다른 원자 모델에 입력 X로 주어진다. 입력 X가 발생할 경우에 원자 모델은 입력에 대한 상태 천이 함수 δ_{ext}를 실행시켜서 상태를 바꾸고 ta 함수를 호출해서 다음 출력이 발생하는 때까지의 시간 간격을 정해준다.

2.2 결합 모델

결합 모델(Coupled Model)은 여러 모델을 내부적으로 연결하여 만든 모델이다. 원자 모델 또는 하위의 결합 모델을 자식으로 가지면서 더 큰 시스템을 표현할 수 있게 된다. 결합 모델의 수학적 명세는 다음과 같다.

$CM = <X, Y, \{M_i\}, EIC, EOC, IC, SELECT>$
X : 이산 사건 입력 집합
Y : 이산 사건 출력 집합
$\{M_i\}$: 모든 이산 사건 콤퍼넌트 모델들의 집합
EIC : 외부 입력 관계
EOC : 외부 출력 관계
IC : 내부 연결 관계
SELECT : $2\{M_i\} - \emptyset \rightarrow M_i$: 같은 시간에 존재하는 사건을 발생하는 모델들에 대한 선택 함수

2.3 계층적인 스케줄링

위에서 말한 DEVS 형식론으로 기술된 모델은 계층적인 알고리즘에 의해서 해석되어 실행된다. DEVS 모델을 실행하는 알고리즘은 두 개의 프로세스로 이루어지는데, 원자 모델을 실행하는 시뮬레이터와 결합모델을 실행하는 코디네이터로 이루어진다[5].

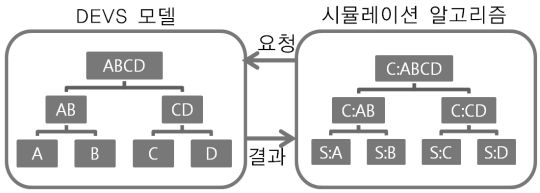


그림 1. 계층적인 스케줄링을 사용한 실행 환경

그림 1은 계층적인 스케줄링을 사용한 실행 환경의 DEVS 모델과 시뮬레이션 알고리즘의 관계를 보여준다. 시뮬레이션 알고리즘은 계층적으로 구현된 DEVS 모델에 대해 1:1로 매핑되어 같은 구조로 이루어진다.

결합모델 ABCD, AB, CD에 대하여 코디네이터(C)가 대응되고 원자 모델 A, B, C, D에 대하여 시뮬레이터(S)가 대응된다. 시뮬레이터와 코디네이터는 DEVS 모델의 함수를 실행하고 모델은 결과를 돌려준다.

예를 들어서 결합모델 ABCD가 있다고 할 때 계층적인 알고리즘에서 A로부터 발생한 메시지가 C로 이동하는 과정은 다음과 같다.

1. 코디네이터 ABCD는 A에 (*)메시지를 보낸다.
2. A의 λ함수에서 (y)를 결합모델AB로 보낸다.
3. AB는 (y)를 다시 결합모델 ABCD로 보낸다.
4. ABCD는 (y)를 (x)로 변환해서 CD로 보낸다.
5. CD는 (x)를 C에 전달한다.
6. C는 (x)를 입력받는 δ_{ext}함수를 실행한다.
7. C는 ta 함수를 실행, 자신의 t_N을 얻는다.
8. C는 CD에게 (done, t_N)을 보낸다.
9. CD는 이미 저장되어 있던 D와 시간을 비교해서 더 작은 t_N을 골라서 ABCD에게 (done, t_N)을 보낸다.
10. A는 계속해서 δ_{int}함수를 실행한다.
11. A는 ta함수를 실행, 자신의 t_N을 얻는다.
12. A는 (done, t_N)을 AB에 보낸다.
13. AB는 자신의 자식 중 가장 작은 t_N 값을 얻어서 다시 ABCD로 (done, t_N)을 보낸다.
14. ABCD는 자신의 자식 중 가장 작은 t_N 값을 얻어서 그 모델을 실행하는 (*)메시지를 전달함으로 시뮬레이션을 계속 진행한다.

※ 각 메시지에 대한 간략한 설명
(*) : 모델의 δ_{int}를 실행하는 메시지이다.
(x) : 모델의 입력 메시지, δ_{ext}를 실행한다.
(y) : 모델의 출력 메시지, λ로부터 발생한다.
(done, t_N) : 자신의 다음 실행 시간을 t_N이라고 하는데 t_N을 부모에게 알리는 메시지이다.

위의 2, 3, 4, 5, 6번은 하나의 원자 모델로부터 나온 출력이 다른 원자 모델에 입력되는 메시지 패싱의 과정이다. 원자 모델 A로부터 나온 출력이 A, AB, ABCD, CD를 거쳐 C에 전달된다. 이 과정에서 커플링 릴레이션(Coupling Relation)이 참고 된다. 커플링 릴레이션이란 한 모델의 출력과 그 다음 경로가 되는 모델의 입력의 릴레이션이다. 2.2장에서 볼 수 있듯이 결합모델에는 EOC, EIC, IC가 있어서 원자 모델과 결합 모델 사이의 릴레이션을 나타낸다.

시간 관리는 하나의 원자 모델이 실행 된 후 다음으로 어느 원자 모델이 실행될지를 결정하기 위해 모델들의 시간을 관리하는 과정이다. 위의 그림에서는 1, 8, 9, 12, 13, 14에서 그 과정을 확인할 수 있다.

계층적인 알고리즘에서는 모델 실행 시간에 대한 전역적인 정보가 존재하지 않고 분산되어 있다. 따라서 원자 모델의 실행 시간이 갱신될 때마다 새로운 시간을 계층적 구조를 통해 수집해야하는 문제가 발생한다. 코디네이터는 자신의 자식들로부터 수정된 시간을 받고 자신의 자식들의 시간 순서를 재배치 한 다음, 그 중에 최소값을 구한다. 결국 최상위 코디네이터는 모든 값을 수집하게 되고 여기서 결정된 최소 실행 시간을 가지는 모델이 실행 명령을 받게 된다. 위의 예에서 실행시간이 수정된 A가(done, tN) 메시지를 AB로 보내고 AB는 다시 ABCD로(done, tN)을 보내는 것을 볼 수 있다.

3. 이벤트 리스트를 이용한 실행 환경

이벤트 리스트를 이용한 실행 환경의 모습은 대략적으로 위와 같이 나타낼 수 있다. 그림 2에는 명세된 DEVS로부터 실행 환경에 맞게 구현된 DEVS 모델과 이벤트 리스트를 사용한 시뮬레이션 엔진이 그려져 있다. DEVS 모델은 계층적인 구조가 없도록 구현되어 있으며 직접 새로운 이벤트를 스케줄 한다. 시뮬레이션 루틴은 이벤트 리스트로부터 새로운 이벤트를 받아서 실행을 명령한다.

3.1 제안하는 실행 환경을 위한 함수 구현

이벤트 리스트를 이용한 실행 환경에서 사용하는 함수

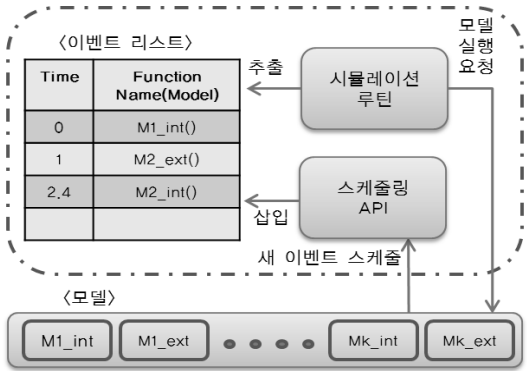


그림 2. 제안하는 실행 환경[6]

그림 3. 제안하는 실행 환경의 외부 천이 함수

그림 4. 제안하는 실행 환경의 내부 천이 함수

는 시스템의 상태 변화를 입력이 있을 때와 입력이 있을 때로 나누어 외부 천이 함수와 내부 천이 함수로 구현하는데, 2.1장에서 살펴본 DEVS의 원자 모델의 실행 순서를 그대로 적용해서 함수를 구현하게 된다.

DEVS는 입력 X가 입력되거나 ta함수가 정한 시간이 다 지나면 모델이 실행되지만 이벤트 리스트를 사용한 환경에서 사용되는 모델은 이벤트에 의해서 상태 천이 함수가 실행된다. 또한 이벤트가 등록되어야 다음 함수가 실행되기 때문에 각 함수는 필요에 따라 이벤트를 등록하는 부분을 포함해야 한다.

그림 3은 제안하는 실행 환경을 위한 외부 천이 함수이다. 이벤트에 의해서 실행된 함수는 입력이 있을 때 실행되는 원자 모델의 함수 순서를 그대로 가진다. 즉, δ_{ext}함수에 해당하는 코드를 실행하여 상태를 바꾸고 ta 함수에 해당하는 코드를 실행하여 다음 모델의 실행 시간까지의 간격을 얻어서 이벤트를 스케줄하게 된다. 반드시 δ_{ext}함수와 ta 함수, 새로운 이벤트를 스케줄 하는 순서로 함수가 구성되어야 하는 것은 아니다. 사용자의 편의에 따라서 자유롭게 구현할 수 있다.

그림 4는 역시 이벤트에 의해서 실행되는 내부 천이 함수 이다. 실행된 함수는 출력을 발생 시키는 λ함수에 해당하는 코드와 입력이 없을 때 상태를 바꾸는 δ_{int} 함수에 해당하는 코드를 실행한다. 그리고 ta함수에 해당하는 코드를 통해 다음 내부 천이 함수의 실행 시간까지의 간격을 얻어서 내부 천이 함수 실행 이벤트를 스케줄하게 된다.

이때 외부 천이 함수 실행 이벤트가 λ함수에 의해서 생성된다. λ함수의 출력은 다른 모델의 외부 천이 함수를 실행시키는 이벤트가 된다. 기존의 계층적인 스케줄링에서는 출력이 계층적인 구조를 통해 전달되어 대상 모델을 실행시키지만 이벤트 리스트를 사용한 실행 환경에서는 이벤트를 통해 전달된다. 즉, 내부 천이 함수에 의해서 발생한 출력은 외부 천이 함수 이벤트에 첨부되며 이 외부 천이 함수 이벤트가 실행될 때는 입력으로서 이벤트에 첨부된 출력을 사용한다.

마찬가지로, 각 코드는 반드시 순서대로 실행되지 않아도 되며, 출력이 없다거나, 내부 천이가 일어나지 않는다거나 하면 해당하는 코드가 없을 수 있다. 이벤트 지향의 실행환경의 모델은 이러한 특성으로 3개의 집합과 4개의 함수를 다 채워야하는 DEVS 형식론을 벗어나 사용자의 판단에 따라 최적화 할 수 있는 가능성을 가진다.

3.2 제안하는 실행 환경의 성능 향상

위와 같은 실행 환경을 사용하면 계층적인 스케줄링으로 인해 발생 하던 시간 관리와 메시지 패싱의 오버헤드를 줄일 수 있다.

2.3장에서 예로 들었던 계층적인 구조에서 A에서 C로 출력이 진행되는 과정은 제안하는 실행 환경에서 다음과 같이 실행된다.

1. 시뮬레이션 엔진은 가장 상위의 이벤트를 추출한다.
2. 이벤트에 의해서 A의 내부 천이 함수가 실행된다 (δ_{int}와 ta 함수가 포함된다.).
3. 내부 천이 함수는 출력이 포함된 외부 천이 함수 실행 이벤트와 내부 천이 함수 실행 이벤트를 추가한다.
4. 시뮬레이션 엔진은 가장 상위의 이벤트를 추출한다.
5. 이벤트에 의해서 C의 외부 천이 함수가 실행된다 (δ_{ext}함수와 ta 함수가 포함된다). 이때 이벤트에 첨부된 메시지가 입력으로 주어진다.
6. 내부 천이 함수 실행 이벤트가 추가된다.
7. 시뮬레이션 엔진은 다음 이벤트를 추출한다.

위의 과정을 2.3장의 내용과 비교하면 다음과 같다.

표 1. 각 과정별 비교

과정	계층적 알고리즘	이벤트 리스트를 사용한 알고리즘
메시지 패싱	2	2
	3,4,5	없음
스케줄링	6,7	5
	1,8,9,12,13,14	1, 4, 7

메시지 패싱과 관련해서 계층적인 구조를 따라 전달되던 작업이 단지 이벤트를 첨부하는 것으로 가능해진다. 또한 계층적인 구조를 통해 실행 시간을 수집하던 시간 관리가 오직 이벤트를 등록하고 이벤트를 추출하는 작업으로 대신 된다. 계층적인 알고리즘의 많은 부분이 삭제되어, 이벤트 리스트를 사용한 실행 환경이 계층적인 스케줄링 알고리즘보다 좋은 성능을 가진다는 결론을 얻을 수 있다.

4. 제안하는 실행 환경의 모델 구현

3장에서 제안하는 스케줄링 알고리즘과 실행 환경을 이루기 위해서 알고리즘에 맞는 이벤트 지향의 모델을 구현해야 한다. 3.2에서 보았듯이 이벤트 리스트를 사용한 실행 환경은 이벤트에 대응되는 2가지 함수가 실행의 단위가 된다. 이 실행의 단위를 가지고 명세된 DEVS를 통해 모델을 만들 때 가장 기본적인 방법으로는 이벤트 지향의 모델 또는 객체 지향의 모델이 있다.

이벤트 지향의 모델 또는 객체 지향의 모델은 기존의 이벤트 리스트의 실행 환경에 맞게 구현되어 있으며 빠른 성능을 가능하게 하지만 동시에 구현하기 불편하다는 문제가 있다.

4.1 이벤트 지향의 모델과 객체 지향의 모델

제안하는 실행 환경을 위한 모델은 다음과 같다.

먼저, 이벤트 지향의 모델은 하나의 이벤트에 하나의 함수가 대응되며 하나의 파일에 이 모든 함수가 전역변수로 누구나 열람할 수 있는 상태 변수(S)들과 함께 저장되어 있다. 따라서 외부 천이 함수는 하나의 모델에 대하여 여러 개가 발생할 수 있는데, 그림 5의 이벤트 지향의 모델을 보면 A_ext1, A_ext2 두 개의 외부 천이 함수가 있는 것을 볼 수 있다. 각각의 함수는 다른 이벤트를 호출하는 기능을 가지고 있으면 화살표로 어떤 이벤트를 호출하는지 표시되어 있다.

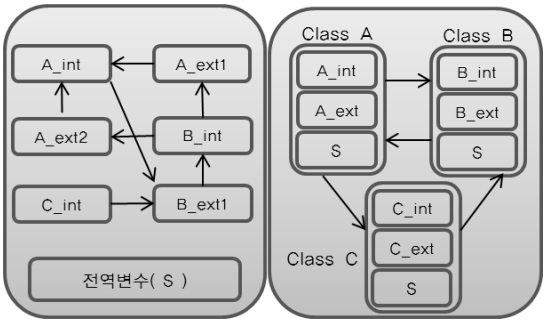


그림 5. 모델 구현의 두 가지 방법

하지만 이러한 이벤트 지향의 모델은 함수로 분리되어 있기 때문에 낮은 재사용성을 가지며 사용자가 구현하기에 불편하다. 따라서 같은 모델에 대해서 하나의 클래스로 묶는 객체 지향의 모델을 생각해 볼 수 있다. 이러한 객체 지향의 모델은 내부 천이 함수와 외부 천이 함수를 가지면서 하나의 클래스 안에 자신의 상태 변수를 가지게 된다. 사용하는 함수의 형태나 실행하는 방법은 같지만 원자 모델이 하나의 클래스로 묶이면서 재사용성이 높아진다.

좀 더 원시적인 방법으로 구현하는 이벤트 지향의 모델이 성능에 있어서 조금 더 빠른 성능을 보일 것이다. 클래스의 멤버 함수를 실행하는 것보다 함수를 실행하는 것이 더 빠르고, 이벤트에 대응하는 함수들이 다 분리되어 있기 때문이다. 하지만 재사용성이라는 측면에서는 객체 지향의 모델이 더 나은 모델 구현 방법이 된다.

4.2 제안하는 실행 환경을 위한 모델의 문제점

앞서 제시한 2가지 모델을 구현함에 있어서는 계층적인 구조가 없다는 것 외에도 구현 편의성에 문제가 있다.

첫째로, DEVS 명세를 구현함에 있어서 DEVS의 4가지 함수를 그대로 구현하지 않고 2개의 함수로 섞어서 구현해야한다. DEVS 명세를 그대로 4개의 함수로 구현할 때보다 더 많은 오류의 가능성을 가지고 있게 된다. 앞서서 말했듯이 2개의 천이 함수를 구현하면서 필요에 따라 최적화 할 수 있는 자유도가 있는데 이러한 자유도는 모델을 구현하고 검증하는데 있어서 가독성, 이해성, 신뢰성을 떨어뜨린다.

둘째로, 계층적인 스케줄링의 실행 환경에서는 모델이 수동적이고 독립적으로 구현되는 반면에 이벤트 리스트

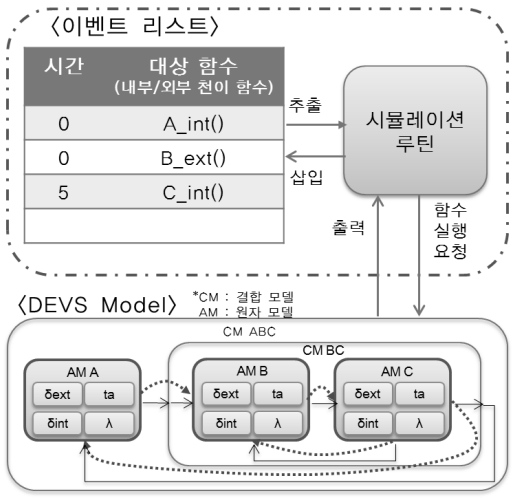


그림 6. 계층적인 DEVS 모델을 사용한 실행 환경[7]

를 사용한 실행 환경에서는 구현된 모델이 직접 다른 모델을 호출하는 이벤트를 스케줄 해야 하기 때문에 시뮬레이션 엔진과 구현된 모델 사이에, 또한 모델과 모델 사이에 완벽한 독립성을 이룰 수 없다.

이러한 두 가지 문제를 해결하기 위해 기존의 계층적이고 모듈러하게 구현된 모델을 이벤트 리스트를 이용하여 스케줄 하는 방법을 제안한다.

5. 계층적 DEVS 모델을 위한 실행 환경

그림 6은 계층적으로 구현된 DEVS 모델을 위한 실행 환경이다. 기존의 이벤트 리스트를 사용한 구현 환경과 가장 다른 점은 계층적으로 구현되어 있고 독립적인 모델이 사용되며 이 모델의 해석을 위해 시뮬레이션 루틴이 더 많은 작업을 한다는 점이다.

이 모델을 사용함으로 성능에 있어서는 이벤트 지향의 실행 환경보다 느리겠지만 사용자가 구현하거나 기존의 모델을 활용하기에 적합한 실행 환경을 구현할 수 있다.

5.1 플래트닝(Flattening)

이벤트 리스트를 사용한 실행 환경에서는 계층적인 구조가 필요 없기 때문에 그러한 구조를 제거하고 시뮬레이션을 시작하는데 이것을 Flattening이라 한다[2]. 중간 경로를 제거하고 원자 모델 끼리를 이어주는 커플링 릴레이션을 만든다. 그림 6에서 실선은 원래 DEVS 모델의 커플링 릴레이션이고 점선은 새롭게 만들어진 커플링 릴레이션으로 원자 모델끼리 연결되어 있다.

표 2. 실험 대상인 4가지 실행 환경

	이벤트 지향 모델의 실행 환경	객체 지향 모델의 실행 환경	DEVSim++ 모델의 실행 환경	DEVSim++ (대조군)
스케줄링 알고리즘	이벤트 리스트	이벤트 리스트	이벤트 리스트	계층적인 알고리즘
이벤트의 실행 대상	이벤트에 대응하는 함수	내부 천이 함수 외부 천이 함수	DEVS의 4개의 함수 (DEVSim++ Model)	DEVS의 4개의 함수 (DEVSim++ Model)
계층적인 구조	X	X	O	O
모델과 엔진의 독립성	약간의 의존성 (모델의 능동적인 속성)	약간의 의존성 (모델의 능동적인 속성)	독립	독립
구현의 편의성	낮은 재사용성 함수 구현에 따른 불편	함수 구현에 따른 불편	비교적 편함 (DEVS의 속성을 그대로 모델에 적용)	비교적 편함 (DEVS의 속성을 그대로 모델에 적용)

5.2 시뮬레이션 루틴의 동작

이벤트 리스트를 사용한 환경에서는 사용되는 내부/외부 천이 함수 내부의 실행되던 코드는 4개의 DEVS 함수를 실행하는 것으로 대신 되어야 한다. 내부 천이 함수는 λ함수, δ_{int}, ta 함수를 실행하도록 하고 외부 천이 함수는 δ_{ext}, ta 함수를 실행해야한다. 이 작업을 시뮬레이션 동작 루틴이 대신해서 실행해준다. 예를 들어, 그림 6에서 이벤트 리스트에 등록되어 있는 A_int()와 B_ext()는 각각 Model A의 λ함수, δ_{int}, ta 함수로, Model B의 δ_{ext}, ta 함수로 대신하여 실행된다.

또한 시뮬레이션 동작 루틴에서 이벤트 등록 또한 대신 해주어야 한다. λ함수에 의해서 반환된 출력을 이용하여 커플링 릴레이션을 참고하여 외부 천이 함수 실행 이벤트를 등록하고 ta함수에 의해서 반환된 시간 값을 이용해 내부 천이 함수 실행 이벤트를 등록한다.

6. 사례 연구

지금까지 이벤트 리스트를 사용하여 스케줄 하는 실행 환경과 그 실행 환경을 위한 3가지 모델 구현 방안을 살펴보았다. 위의 실행 환경들의 정확성 검증과 성능 테스트를 위해서 각각의 실행 환경을 C++로 구현하고 명세된 DEVS를 각각의 실행 환경에 맞는 모델로 구현하였다. 성능 비교를 위해서 각 구현물에서 공유할 수 있는 부분은 최대한 공유하도록 하였으며, 이벤트 리스트를 관리하는 힙 구조 또한 기존의 DEVSim++ 모델이 모델 중에 가장 빠른 t_N을 가지는 모델을 결정할 때 사용하는 힙 구조를 그대로 사용하였다.

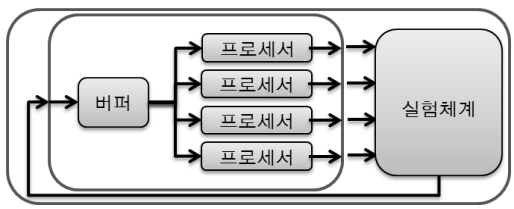


그림 7. 단일 서버 큐잉 모델

6.1 실험 환경

실험에는 표 2와 같이 3개의 실행 환경을 구현하고 대조군으로서 계층적인 스케줄링을 사용한 DEVSim++을 사용하였다. DEVSim++은 DEVS 형식론을 C++으로 구현한 실행 환경이다[8]. 따라서 계층적으로 구현된 DEVS 모델 또한 DEVSim++을 위해 구현된 모델을 사용하였다. 구현의 편의성이라는 측면에서, 명세된 DEVS에 가깝다는 면에서 오른쪽으로 갈수록 장점이 있다. 하지만 성능은 왼쪽으로 갈수록 빠른 성능을 보일 것으로 예상된다. 실험에는 단일 서버 큐잉 모델을 사용했으며 그림 7과 같다.

실험 체계(Experimental Frame)에서 생성된 메시지가 버퍼(Buffer)와 프로세서(Processor)를 거쳐 다시 실험체계에 전달되며 깊이가 2인 계층적인 구조를 가지고 있으며 약 7개의 원자 모델이 활성화 되어 동작한다.

DEVS로 명세된 위의 모델을 표2의 실행 환경에 맞게 각각 실행 환경에 맞게 구현하여 비교, 실험 하였다.

6.2 정확성 검증

동작의 정확성 검증을 위해서 4개의 실행 환경에 같은 입력 시퀀스를 넣었다. 입력 시퀀스는 4개의 모델에 같은 랜덤 값이 입력되도록 하였다. 그 결과로 나오는 출력의

표 3. 정확성 검증

실행 환경	출력
이벤트 지향 모델의 실행 환경	DEVSim++와 같음
객체 지향 모델의 실행 환경	DEVSim++와 같음
DEVSim++ 모델의 실행 환경	DEVSim++와 같음

순서를 확인해서 대조군과 같은 결과가 나오는지 확인하였다.

표 3의 결과를 통해서 기존의 검증되어 있는 실행 환경인 DEVSim++과 모두 같은 결과를 보인다는 확인하였다.

6.3 성능 측정 및 비교

성능 측정을 위해서 한 번에 활성화 되는 모델의 개수를 늘리면서 실행 했다. 대조군으로서 깊이가 1이 되도록 계층적 구조를 수정한 모델을 사용하였다. 이벤트 리스트를 사용한 환경에서는 계층적인 구조가 없기 때문에 계층적인 모델을 사용하는 것이 불공평할 수 있다. 따라서 똑같은 모델이지만 계층 깊이가 1이 되도록 수정하여 실행했다.

그림 8은 성능 측정의 결과이다. 점선은 대조군인 DEVSim++ 실행 환경의 결과이고 나머지 3개의 실선은 순서대로 DEVSim++ 모델 - 객체 지향 모델 - 이벤트 지향 모델의 그래프이다. 대조군을 기준으로 해서 몇 배 성능 향상이 있었는지를 간단히 표로 나타면 다음과 같다.

표 4를 통해서 이벤트 리스트를 사용한 실행 환경들이 모두 빠른 성능을 보였으며 그 성능은 이벤트 지향의 모델 - 객체 지향 모델 - DEVSim++ 모델 순이라는 것을 확인할 수 있다. 이 결과는 구현 편의성과 정 반대되는 결과로서 장단점이 있다는 것을 확인할 수 있다.

이벤트 지향 모델은 함수단위로 이루어져 한 파일 안에 존재하기 때문에 클래스의 멤버 함수를 호출하는 객체 지향 모델보다 성능이 좋다. DEVSim++ 모델을 사용할 때는 모델을 해석하고 실행하는 부분이 필요하기 때문에 객체 지향 모델을 사용하는 환경이 DEVSim++ 모델을 사용하는 환경보다 빠른 성능을 보인다.

위의 실험 결과는 이벤트 지향 DEVS 실행 환경의 성능 향상을 주장하기에 두 가지 부족한 점이 있다.

첫째로, 성능은 공간을 비효율적으로 더 사용함으로써 좋아질 수 있기 때문에 효율적인 공간 사용이 이루어졌는

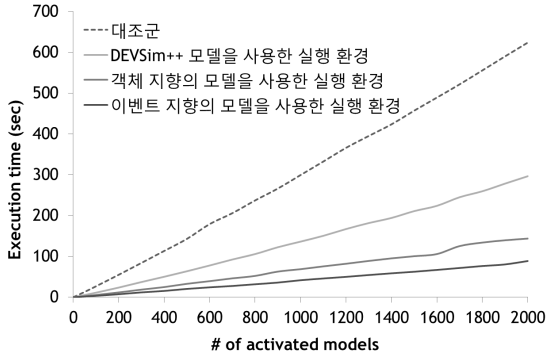


그림 8. 성능 측정 결과

표 4. 성능 향상(모델 2000개 기준)

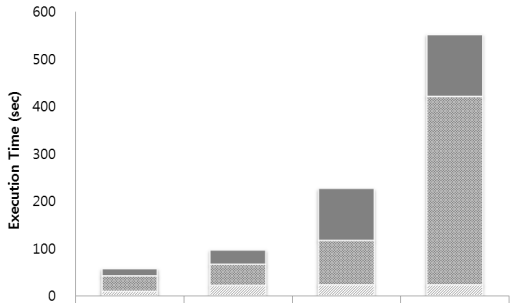
이벤트 리스트를 사용한 실행 환경	대조군 대비 성능 향상
이벤트 지향 모델의 실행 환경	약 7배
객체 지향 모델의 실행 환경	약 4.3배
DEVSim++ 모델의 실행 환경	약 2.1 배

지 확인이 필요하다. 크게 봤을 때, 대조군에 비해서 이벤트 지향 실행 환경은 이벤트를 저장하기 위한 공간을 더 사용한다. 하지만 이벤트의 수는 최대 모델의 개수와 동시에 발생하는 모델의 출력의 개수를 합친 것을 넘지 않기 때문에 메모리 사용의 심각한 증가를 초래하지 않으며, 이벤트 개수가 증가함에 따라 선형으로 증가한다. 따라서 성능 개선을 위해 공간을 효율적으로 사용함을 알 수 있다.

둘째로, 실험 결과에는 모델 자체의 실행 시간도 들어 있기 때문에 성능 개선이 스케줄링 알고리즘에 따른 것인지 확인하는 것이 필요하다. 따라서 프로파일링(Profiling)을 통해 알고리즘의 개선에 따른 성능 향상의 정도를 확인했다.

그림 9는 모델의 개수가 2000개일 때 실행 시간을 프로파일링 하여, 메시지 패싱, 시간 관리, 모델 실행에 대하여 실행 시간을 정리한 그래프이다.

이 그림을 통해 모델 자체의 실행 시간에 관계없이 계층적인 알고리즘을 사용하는 대조군의 시간 관리와 메시지 패싱의 오버헤드를 줄인 것을 볼 수 있으며, 앞선 그림 8의 실험 그래프가 실제로 계층적인 오버헤드를 줄인 결과라는 것을 확인 할 수 있다.



	이벤트 지향의 모델을 사용한 실행 환경	객체 지향의 모델을 사용한 실행 환경	DEVSim++ 모델을 사용한 실행 환경	대조군
메시지 패싱	15.68	30.89	109.70	130.64
시간 관리	32.44	44.39	93.74	399.40
모델 실행	9.04	21.34	22.94	22.88

그림 9. 모델의 개수가 2000개일 때 프로파일링 결과

7. 결 론

지금까지 DEVS 실행 속도를 개선하기 위한 이벤트 리스트의 실행 환경들을 제안하고 성능을 살펴보았다.

계층적 구조에 따른 문제점을 가지고 있는 계층적 알고리즘 대신에 이벤트 리스트를 사용하여 이벤트 관점으로 스케줄링 함으로써 성능 향상을 얻을 수 있었다. 실행 환경의 모델은 3가지 방법으로 구현 할 수 있는데 이벤트 지향 - 객체 지향 - 기존의 계층적인 모델 순으로 속도가 빠르지만 그만큼 모델 구현에 있어서 단점이 있다.

따라서 필요한 성능에 따라 적합한 실행 환경을 선택해야하며, 극단적인 실행 환경의 속도가 필요한 경우를 제외하고 기존의 계층적으로 구현된 모델을 사용하면서 이벤트 리스트로 스케줄링 하는 것이 가장 합리적인 선택으로 보인다.

감사의 글

본 연구는 문화체육관광부 및 한국콘텐츠진흥원의 2009년도 문화콘텐츠산업기술지원사업의 연구결과로 수행되었음.

참 고 문 헌

1. Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim, *Theory of modeling and simulation*. ACADEMIC PRESS, 2001.
2. Y.G. Kim and T.G. Kim, “Optimization of model execution time in the DEVSim++ environment,” *In proc. of 1997 european simulation symposium*, Passau, Germany, pp. 215-219, Oct., 1997.
3. Wan Bok LEE and Tag Gon KIM, “Performance evaluation of concurrent system using formal model: simulation speedup,” *IEICE Transactions on fundamentals of electronics*, Communications and computer sciences, Vol. E86-A, No. 11, pp. 2755-2766, 2003.
4. Tag Gon KIM, IE801 Lecture Note, EE, KAIST, 2009, http://smslab.kaist.ac.kr/
5. Tag Gon KIM, *DEVSim++ v3.0 Developer’s manual*, 2006, http://smslab.kaist.ac.kr/
6. 권세중, 김탁곤, “Event-oriented 기반의 DEVS 모델 실행 환경,” 한국군사과학기술학회 '10종합학술대회 , pp. 380-383, 2010년 6월.
7. 권세중, 김탁곤, “계층적인 DEVS 모델의 이벤트 기반 실행 환경,” 한국시뮬레이션학회 '10추계학술대회, pp. 117-122, 2010년 10월.
8. Tag Gon Kim and Sung Bong Park, “The DEVS formalism: hierarchical modular systems specification in C++,” *1992 European simulation multiconference*, York, United Kingdom, pp. 152-156, June, 1992.

권 세 중 (sjkwon@smslab.kaist.ac.kr)

2009 한국과학기술원 전산학과 학사
2011 한국과학기술원 전기및전자공학과 석사
2011～현재 한국과학기술원 전기및전자공학과 박사과정

관심분야 : 모델링&시뮬레이션, DEVS 형식론, 시뮬레이션 엔진

김 탁 곤 (tkim@ee.kaist.ac.kr)

1975 부산대학교 전자공학과 학사
1980 경북대학교 전자공학과 석사
1988 Univ. of. Arizona, 전기및컴퓨터공학과 박사
1980～1983년 부경대학교, 통신공학과, 전임강사
1987～1989년 (미)아리조나 환경연구소, 연구엔지니어
1989～1991년 Univ. of Kansas, 전기및컴퓨터공학과, 조교수
1991～현재 KAIST 전자전산학과, 교수
- 한국시뮬레이션 학회 회장 역임
- 국제시뮬레이션학회(SCS) 논문지(Simulation) Editor-In-Chief 역임
- SCS Fellow
- 모델링 시뮬레이션 기술사(미국)
- Who's Who in the World(Marguis 16thEdition, 1999) 등재
- 연합사, 국방부/합참, 기품원 자문위원 역임
- KIDA Fellow 역임
- ADD 자문위원(현)

· 연구 실적
- 교재
  Theory of Modeling and Simulation, Academic Press, 2000, 등 영어 교재/Chapter 8권 저술
- 연구 논문
  국·내외 학술지/학술대회 M&S 관련 논문 200여 편 발표
- 과제 수행
  훈련, 분석, 획득, 전투실험 관련 국방 M&S 과제 10여 건 수행(중)

관심분야 : 모델링/시뮬레이션 이론, 방법론 및 환경개발, 시뮬레이터 연동