계층적인 DEVS 모델의 이벤트 기반 실행 환경

권세중, 김탁곤*

Event-oriented Execution Environment For Hierarchical DEVS Model

Se Jung Kwon and Tag Gon Kim

Abstract

DEVS(Discrete Event Systems Specification) is a set theoretic formalism developed for specifying discrete event systems. For the DEVS model execution, we need a scheduling algorithm in an execution environment. An existing algorithm of DEVS is a hierarchical scheduling algorithm, which has overheads owing to message passing and execution time management. To overcome these overheads, there is event-oriented execution environment. But hierarchical structure of DEVS have to be modified for event-oriented approach. Some errors must be occurred from the modification. So there is need about event-oriented execution environment without modifying DEVS. This paper proposes improved event-oriented execution environment using hierarchical DEVS model. We experiment about the performance of our proposed environment in comparison with the existing execution environment using the hierarchical scheduling. The experimental result show proposed environment works better than existing environment using same DEVS models.

Key Words : Event-oriented, DEVS, Discrete Event System, Simulation Speedup

* 한국과학기술원 전기및전자공학과

1. 서론

이산 사건 시스템이란 실수 시간에 일어나는 이벤트에 의해서 상태가 변화하는 시스템이다. 이러한 이산 사건 시스템은 통신 모델, 생산 시스템, 워 게임 등에서 찾아 볼 수 있다.

이러한 이산 사건 시스템의 행동이나 성능을 분석하기 위해서는 모델링을 통해 모델을 만든다. 모델링을 위해서는 수학적인 틀(Framework)을 활용하는 방법이 많이 쓰이는데, 이산사건 시스템의 대표적인 수학적인 틀에는 이산사건 시스템 명세(Discrete Event System Specification, DEVS)가 있다[1].

DEVS의 정형적이고 계층적인 틀을 통해 구현된 모델은 주로 계층적인 스케줄링 알고리즘을 통해 실행된다. 이러한 알고리즘은 정형적인 DEVS 모델의 구조를 적용하여 시뮬레이터를 구현한다. 각 모델에 대해 일대일로 대응되는 시뮬레이터(Simulator)와 코디네이터(Coordinator)가 존재한다. 이들이 DEVS 모델의 함수를 호출하고 결과를 돌려받으면서 시뮬레이션이 진행된다.

이러한 계층적인 스케줄링 방식은 DEVS 모델과 알고리즘이 같은 계층적 구조를 가진다는 장점이 있다. 대신 계층적 구조로 인해 입출력 및 스케줄링 메시지가 이동하는 과정 가운데 오버헤드가 발생한다. 이 문제를 해결하기 위해서 각 원자모델들을 하나의 모델로 합성하려는 연구가 이뤄진바 있으나 이 방법은 기존의 모델을 수정한 뒤에 실행을 하기 때문에 디버깅이 어렵다는 문제가 있다[2].

최근에는 이벤트 지향의 실행 환경을 통한 성능 향상에 대한 연구가 된 바 있다[3]. 이벤트 지향의 실행 환경은 모델의 실행이 이벤트에 의해서 이루어진다. 이벤트 리스트에 등록된 이벤트에 의해서 모델이 실행되고 실행된 모델은 다시 이벤트를 발생시켜 이벤트 리스트에 등록한다. 시뮬레이션 동작 루틴은 이 작업을 반복함으로 시뮬레이션이 진행되도록 한다. 이벤트 지향의 DEVS 모델은 외부 천이 함수와 내부 천이 함수로 이루어지고 계층적인 구조가 없다. 따라서 계층적인 구조를 그대로 구현한 DEVS 모델은 이벤트 지향의 실행 환경에서 실행하기 위해 수정되어야 한다. 이렇게 사용자에 의해 수정된 모델은 본래 기술된 DEVS와 비교했을 때 오류가 있을 가능성이 높아진다는 문제점이 있다.

따라서 본 논문은 이벤트지향의 실행 환경을 확장하여 기존의 계층적 실행 환경에서 쓰이는 계층적인 DEVS 모델을 사용할 수 있도록 한다. 본 논문이 제안하는 실행 환경은 기존의 구현된 DEVS 모델의 인터페이스를 유지하기 때문에 기존의 모델을 그대로 사용할 수 있고 사용자의 노력이 더 필요하지 않다.

본 논문의 구성은 다음과 같다. 2장에서 계층적인 스케줄링 알고리즘에 대해서 살펴보고 3장에서는 이벤트 관점에서 접근하지만 계층적으로 구현된 DEVS 모델을 그대로 실행하는 환경을 제안하며 이론적인 성능을 분석한다. 4장에서는 단일 서버 큐잉 모델을 통해 성능을 실험하고 비교, 분석한다. 그리고 5장에서 결론을 맺는다.

2. DEVS와 계층적인 스케줄링 알고리즘

DEVS 형식론은 이산 사건 시스템을 객체 단위로 모듈화 하여 계층적으로 결합하여 표현하는 집합론에 근거한 수학적인 틀이다. 시스템의 기본적인 구성단위를 나타내는 원자 모델과 여러 모델을 합쳐서 새로운 모델을 구성할 수 있는 결합 모델이 있다[1]. 원자 모델은 3개의 집합과 4개의 함수로 구현 되어 있다. DEVS 모델 실행환경은 모델의 4개의 함

수를 호출시켜 모델을 실행한다. DEVS의 4개의 함수는 다음과 같다.

- δ_{ext} : 외부의 입력에 따라 모델의 상태를 바꾼다.
- δ_{int} : 일정시간동안 입력이 없을 때 상태를 바꾼다.
- λ : 일정 시간동안 입력이 없을 때 출력을 내보낸다.
- ta : 모델의 다음 실행 시간을 알려준다.

DEVS모델을 실행하는 계층적인 시뮬레이터 알고리즘은 두 개의 프로세스로 이루어지는데, 원자모델을 실행하는 시뮬레이터와 결합모델을 실행하는 코디네이터로 이루어진다[4].

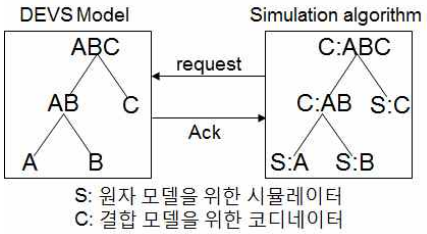

그림 1 계층적인 시뮬레이션 구조[4]

그림 1은 DEVS 모델과 시뮬레이션 알고리즘의 관계를 보여준다. 결합모델 ABC, AB에 대해서 코디네이터(C)가 대응되고 원자 모델 A, B, C에 대해서 시뮬레이터(S)가 각각 대응된다. 시뮬레이터와 코디네이터는 DEVS 모델의 함수를 실행하고 모델은 결과를 돌려준다.

계층적 스케줄링 알고리즘을 통해 모델을 실행하면 메시지 패싱과 스케줄링의 과정에서 계층적인 구조로 인한 오버헤드가 발생한다.

예를 들어 그림 1과 같은 결합 모델 ABC가 있다고 할 때 계층적인 알고리즘에서 A로부터 발생한 메시지가 C로 이동하는 과정은 다음과 같다.

1. ABC는 A에 (*) 메시지를 보낸다.
2. A의 λ함수에서 (y)를 결합모델AB로 보낸다.
3. AB는 (y)를 다시 결합모델 ABC로 보낸다.
4. ABC는 (y)를 (x)로 변환해서 C로 보낸다.
5. C는 (x)를 입력받는 δ_{ext}함수를 실행한다.
6. C는 ta 함수를 실행, 자신의 t_N을 얻는다.
7. C는 ABC에게 (done, t_N)을 보낸다.
8. A는 계속해서 δ_{int}함수를 실행한다.
9. A는 ta함수를 실행, 자신의 tN을 얻는다.
10. A는 (done, t_N)을 AB에 보낸다.
11. AB는 자신의 자식 중 가장 작은 t_N 값을 얻어서 다시 ABC로 (done, t_N)을 보낸다.
12. ABC는 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.

메시지 패싱은 하나의 원자모델에서 출력이 발생 할 때 이 메시지가 다른 원자모델에 입력되기까지 메시지를 전달하는 과정이다. 위의 예에서 원자 모델 A로부터 메시지가 출력되면 모델 A, AB, ABC를 차례로 통과하면서 C에 도달한다(2, 3, 4, 5의 과정). 이 과정에서 커플링 릴레이션이 참조된다. 커플링 릴레이션이란 한 모델의 출력과 그 다음 경로가 되는 모델의 입력의 릴레이션이다. 원자모델로부터 출력이 생기게 되면 커플링 릴레이션을 따라서 모델들을 거쳐 원자모델에 전달된다.

스케줄링은 하나의 원자모델이 실행된 후 다음으로 어느 원자모델이 실행될지를 결정하는 과정이다. 계층적인 알고리즘에서는 모델 실행 시간에 대한 전역적인 정보가 존재하지 않는다. 따라서 원자모델들의 실행시간이 매번 갱신될 때마다 갱신된 실행 시간을 계층적 구조를 통해 수집해야 하는 문제가 있다. 코디네이터는 다음 실행할 모델을 결정하기 위해서 자신의 자식 중에서 가장 작은 값을 골라야 하고 이를 위해서 자식들로부터 수정된 실행 시간을 받는다. 위의 예에서 실행 시간이 수정된 A가 (done, t_N) 메시지를 AB로 보내고(10), AB는 ABC로 (done, t_N)을 보내는 것(11)을 볼 수 있다.

3. 제안하는 실행 환경

제안하는 실행환경의 기본적인 작동 방식은 이벤트 지향의 실행 환경과 같다. 다만, 이벤트 지향적인 DEVS 모델을 실행하는 것이 아니라 계층적으로 구현된 기존의 모델을 불러서 변환해서 사용한다는 점에서 다르다. 그림 2는 계층적인 DEVS 모델을 위한 이벤트 지향의 실행 환경이다.

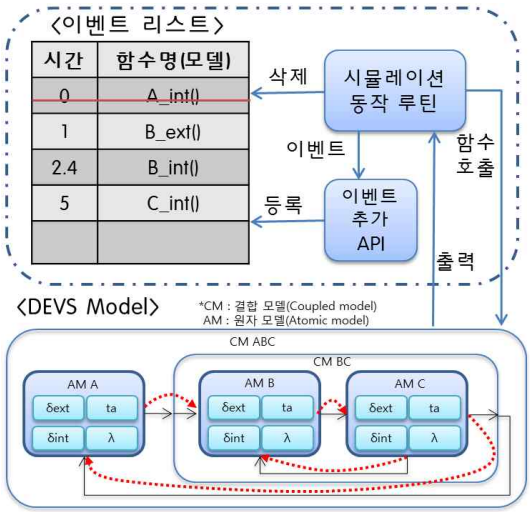

그림 2 제안하는 실행 환경

3.1 Flattening

제안하는 실행 환경에서는 계층적인 스케줄링을 하지 않기 때문에 계층적인 구조를 없애는 데, 이 작업을 Flattening이라고 한다. 중간 경로를 삭제하고 처음 출력이 나가는 모델과 마지막으로 입력이 들어가는 모델간의 커플링 릴레이션을 구한다.

그림 2의 DEVS 모델에서 실선은 원래 존재하는 커플링 릴레이션이고 점선은 결합모델을 무시한 원자모델간의 커플링 릴레이션이다. 시뮬레이션이 시작되면 Flattening을 통해 만들어진 새로운 커플링 릴레이션을 사용해서 중간 과정 없이 바로 메시지를 보낸다.

3.2 제안하는 실행 환경의 동작

이러한 DEVS 모델 실행 환경은 시뮬레이션 동작 루틴의 모델 실행 알고리즘이 조금 다르다. 이벤트 지향 실행 환경에서는 그림 2의 이벤트가 가리키는 A_int 함수가 구현되어 있다. 즉, λ함수, δ_{int}함수 그리고 ta함수의 결합된 형태의 하나의 함수 A_int 함수가 존재하고 이벤트의 지시에 따라 실행된다. 하지만 제안하는 실행 환경은 4개의 함수가 그대로 남아 있는 모델이므로 이벤트가 가리키는 함수의 내용을 시뮬레이션 동작 루틴이 대신 실행한다. 즉, 내부 천이 함수 이벤트가 발생한다면 λ함수, δ_{int}함수 그리고 ta함수를 차례대로 발생시키고 외부 천이 함수 이벤트가 발생한다면 δ_{ext}함수와 ta함수를 발생시킨다.

또한 시뮬레이션 동작 루틴에서 이벤트 등록 작업을 대신 해주어야 한다. δ_{ext}함수 또는 δ_{int}함수가 끝났을 때 ta함수를 호출해서 다음 내부 천이 함수의 실행 시간을 얻고 내부 천이 함수 이벤트를 등록한다. 또한 λ함수가 끝났을 때는 출력의 종착지를 원자모델간의 커플링 릴레이션을 통해 확인해서 그 출력에 대한 이벤트를 등록한다. 이러한 작업이 위에서 말한 다음 실행하는 모델과 시간을 결정하는 스케줄링 작업이다.

또한 λ함수에 의해서 만들어진 출력을 목적지까지 이동시키는 작업(메시지 패싱)이 필요한데 이것은 이벤트를 통해서 이루어진다. 동작 루틴은 λ함수가 출력한 메시지를 등록되는 외부 천이 함수 이벤트에 첨부하고 해당 이벤트가 실행될 때 첨부된 출력을 받아서 그대로 δ_{ext}함수에 입력으로 넣는다.

3.3 시뮬레이션 오버헤드 분석[1)]

2장에서 그림 1을 통해 다룬 예제는 이벤트

1) 3.3장의 분석방법은 [2]의 논문에서 결합된 모델의 성능 분석에 사용되었다.

지향의 알고리즘에서 다음과 같이 실행된다.

1. 이벤트에 의해서 A의 λ함수가 실행되고 출력과 함께 이벤트가 발생한다.
2. A의 δ_{int}함수를 실행한다.
3. A의 ta함수를 실행 후 그 값으로 다음으로 실행될 A의 이벤트를 발생시킨다.
4. 이벤트로부터 C의 δ_{ext}함수를 실행한다.
5. C의 ta함수를 실행하고 그 값으로 다음으로 실행될 A의 이벤트를 발생시킨다.
6. 이벤트 리스트로부터 이벤트를 얻어 계속해서 시뮬레이션을 진행한다.

※ 이벤트의 등록하고 삽입하는 오버헤드는 별도로 주어진다. 일반적으로, 삽입하고 삭제하는 과정이 이벤트가 길어짐에 따라 느려진다. 이로 인해 발생하는 문제는 4장 실험에서 다룰 것이다.

표 1 각 과정별 비교

과정	계층적 알고리즘	이벤트지향 알고리즘
메시지 패싱	2	1
	3,4	없음
스케줄링	6	5
	9	3
	1,7,10,11,12	없음

표 1에서 위의 내용을 각 과정별로 나누어서 비교 정리했다. 계층적 알고리즘의 2, 3, 4를 거치는 메시지 패싱 과정이 이벤트 지향 알고리즘에서는 한 번의 이벤트 등록으로 마쳐지며 스케줄링 과정 또한 done 메시지를 보내는 과정(7, 10, 11)이나 실행을 명령하는 (*) 메시지(1, 12)없이 바로 이벤트를 등록하고 실행하는 것을 볼 수 있다.

4. 실험 및 분석

세 가지 목적을 가지고 실험을 계획하였다.

첫 번째, 위에서 분석한대로 제안하는 실행 환경이 계층적 DEVS 모델 실행에 있어서 성능 향상을 가져온다는 것을 보인다.

두 번째, 범위성(Scalability)에 대한 것이다. 이벤트의 개수가 많아질수록 삽입 및 삭제 속도가 전체 실행 시간에서 차지하는 비중이 커지고 성능이 저하되기 때문에 활성화된 모델의 수가 늘어남에 따라 제안하는 실행 환경의 성능 변화를 살펴본다.

세 번째, 계층적 스케줄링을 사용한 실행 환경은 깊이가 깊어질수록 계층적 구조로 인한 오버헤드가 커진다. 그리하여 똑같은 기능을 하는 모델을 깊이만 다르게 해서 기존의 실행 환경으로 실행한다. 이벤트 지향의 제안하는 실행 환경은 Flattening을 통해 계층을 제거하기 때문에 깊이가 깊어져도 같은 성능을 보일 것이나 기존의 계층적인 실행 환경은 깊이가 깊어질수록 성능이 저하될 것이다.

비교 대상이 되는 계층적인 DEVS 모델 실행 환경으로 DEVS 형식론을 C++ 언어로 구현한 DEVSim++를 사용하였다[5]. 제안하는 실행 환경과 DEVSim++의 비교 실험을 위해서 동일하게 구현된 단일서버 큐잉 모델로 실험했다. 모델의 대략적인 그림은 다음과 같다.

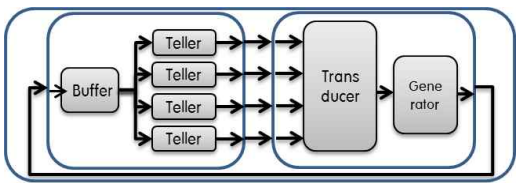

그림 3 단일 서버 큐잉 모델

Generator에서 생성된 메시지가 Buffer와 Teller를 거쳐 Transducer로 전달되며 깊이가 2인 계층적인 구조를 가지고 있으며 약 7개의 원자모델이 활성화 되어 동작한다.

만개의 메시지를 처리하는 단일 서버 큐잉 모델을 1개부터 1500개까지 동시에 실행하면서 실험을 하였으며 단일 서버 큐잉 모델의 깊이를 증가시키면서 실험을 반복하였다. 위의 모델의 깊이를 늘리기 위해 그림 4와 같은 구조를 사용하였다.

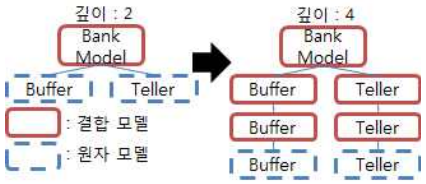

그림 4 깊이를 늘리는 방법

그림 4는 그림 3 모델의 일부분을 예를 보이기 위해 계층적 구조로 그린 것인데, 깊이를 늘리기 위해서 원자모델 위에 결합모델을 추가하여 깊이를 늘린다. 이 같은 방법으로 Generator, Buffer, Teller의 깊이를 2, 3, 4, 6으로 늘려서 실험하였다. 모델의 행동은 같지만 계층적인 구조에 따른 오버헤드는 증가할 것이다. 그 결과는 아래와 같다.

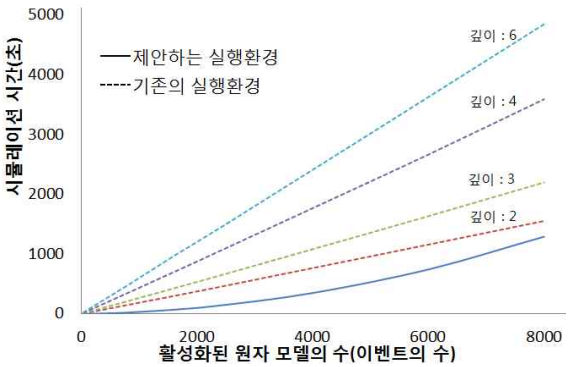

그림 5 성능 비교

그림 5의 실선은 제안하는 실행 환경의 결과이며, 4개의 점선은 기존의 실행 환경이다. 이벤트 기반의 실행환경은 기존의 실행환경보다 더 좋은 성능을 보인다.

또한 이벤트가 많아짐에 따라 제안하는 실행환경의 성능 개선 효과가 점차 사라지지만 모델의 깊이가 깊어질수록 기존의 실행환경의 성능이 떨어지기 때문에 제안하는 실행 환경이 여전히 장점을 가지는 것을 볼 수 있다.

5. 결론

본 논문은 DEVS 모델의 성능을 개선하는 방법으로 계층적인 모델을 수정하지 않고 그대로 사용하면서도 스케줄링과 메시지패싱 오버헤드를 개선하는 방법을 제안했고 기존의 실행 환경인 DEVSim++와 같은 모델 구현물로 성능을 비교하였다.

현실적으로 동시에 활성화되는 모델의 개수가 1000개를 넘기 힘들다는 것을 감안하고 DEVS 모델의 깊이가 깊어질수록 기존의 실행 환경의 성능 저하를 생각한다면 제안하는 실행 환경은 기존의 계층적 구조 실행 환경의 좋은 대안이 될 것이다.

감사의 글

본 연구는 문화체육관광부 및 한국콘텐츠진흥원의 2009년도 문화콘텐츠산업기술지원사업의 연구결과로 수행되었음

참고문헌

[1] Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim, *Theory of Modeling and Simulation*. ACADEMIC PRESS, 2001.

[2] 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

[3] 권세중, 김탁곤, "Event-oriented 기반의 DEVS 모델 실행 환경," 한국군사과학기술학회 '10종합학술대회 , pp. 380 - 383, 2010년 6월

[4] Tag Gon KIM, DEVSim++ v3.0 Developer's Manual, 2006, http://smslab.kaist.ac.kr/.

[5] 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.