Event-oriented 기반의 DEVS 모델 실행 환경
DEVS Model Execution Environment Based on Event-oriented Approach

권세중 김탁곤
Se Jung Kwon and Tag Gon Kim
한국과학기술원 전기 및 전자공학과
(발표자 연락처 : 042-350-5454, sjkwon@smslab.kaist.ac.kr)

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. As the scheduling algorithm determines the execution order of DEVS model, it makes an effect on the execution environment. An existing algorithm of DEVS is a hierarchical scheduling algorithm. But, this algorithm has overheads owing to message passing and execution time management. Therefore, we propose DEVS execution environment using event-oriented approach to improve the execution time. In our environment, the scheduling of model execution is performed by events and an event list. We experiment about the performance of our proposed environment in comparison with the existing execution environment using the hierarchical scheduling. The experimental result show event-oriented approach works better than existing environment using the hierarchical scheduling.

Keywords : DEVS Formalism; DES Simulation; Event-oriented; Simulation Speedup

1. 서 론

이산 사건 시스템 시뮬레이션은 국방 분야에서 시스템 분석 및 훈련 도구로 널리 사용되어 왔다. 훈련을 위한 워게임 모델, 전력 임무급/전구급 분석, 전투체계 획득 및 효과도 분석을 위한 교전급 모델을 예로 들 수 있다. 이러한 이산 사건 시스템을 모델링 하는 방법에는 수학적 공식을 통한 접근법과 관점적 접근법(World-view)이 있다.

수학적 공식을 통한 접근법은 수학적으로 표현하는 틀(Framework)을 제공한다. 수학적 공식을 통한 접근 방법은 구현하고자 하는 모델의 내용을 채워 넣을 정형적인 틀을 제공하기 때문에 모델링이 편하고 표현력이 좋으며 자동 검증에 용이하다. 대표적인 이산 사건 모델의 수학적인 공식에는 이산 사건 시스템 명세(Discrete Event Systems Specification, DEVS) 형식론이 있다 [1]. DEVS는 3개의 집합과 4개의 함수로 수학적인 틀을 제공하며 시스템을 모듈화 하고 계층적으로 결합하여 표현한다.

그러나 이런 DEVS모델을 실행하기 위한 계층적 스케줄링 알고리즘은 정형적인 틀을 엄격하게 적용하여 모델을 스케줄링 하기 때문에 실행 속도가 저하된다. 따라서 성능의 개선을 위하여 모델의 구조를 바꾸거나 모델을 합성하는 등의 노력이 있었다 [2][3].

반면, 관점적 접근법은 모델링 하는 사람이 보는 관점에 따라 모델을 프로그램 언어적으로 표현하며 모델링보다 모델 실행에 초점을 맞추고 있다. 이러한 접근법은 정형적이지 않은 모델 구현으로 인해 자동 검증이 어려운 대신 모델을 실행 하는데 있어서 모델이 프로그램에 가깝기 때문에 스케줄링 알고리즘이 간단하고 성능이 좋다.

본 논문은 DEVS 모델의 실행속도를 개선하기 위해 DEVS 모델을 Event-oriented 접근법에 기반한 실행 환경에서 실행한다. Event-oriented 기반의 이산 사건 시뮬레이션은 실행이 이벤트리스트와 이벤트에 의해서 이루어진다. 이벤트리스트에 등록된 이벤트에 의해서 모델이 실행되고 그 모델은 다시 이벤트를 발생시켜 이벤트리스트에 등록한다. 시뮬레이션 동작 루틴이 반복해서 이벤트리스트에 저장된 이벤트를 실행시킴으로 시뮬레이션을 진행한다.

본 논문은 이벤트 리스트를 통해 DEVS 모델의 실행을 스케줄링 하는 Event-oriented DEVS 모델 실행 환경을 제안하고 기존의 계층적 스케줄링에 기반한 실행 환경보다 나은 성능을 가짐을 보인다. 또한, Event-oriented 실행 환경의 구현방안에 대해 논한다. 이러한 실행 환경을 구현함에 있어 고려해야 할 사항은 이벤트 리스트의 길이가 늘어남에 따른 성능변화이다. 따라서 리스트의 자료구조의 구현방식이 중요하며, 본 논문에서는 이벤트리스트의 자료구조 구현에 따른 성능에 대해서도 분석한다.

본 논문의 구성은 다음과 같다. 2장에서 DEVS 모델과 DEVS 모델의 계층적 스케줄링 알고리즘에 대해 간단히 살펴본 후 3장에서 Event-oriented 기반의 DEVS 모델 실행 환경을 제안한다. 4장에서는 Event-oriented DEVS 모델 실행 환경의 설계 및 구현에 대해서 논한다. 5장에서는 단일 서버 큐잉 모델을 이용하여 Event-oriented 기반의 모델 실행 환경의 성능을 시험하고 계층적인 스케줄링을 통해 DEVS 형식론 모델을 실행하는 기존의 실행 환경과 비교한다. 그리고 마지막으로 6장에서 결론을 맺는다.

2. DEVS 형식론과 계층적 스케줄링 알고리즘

DEVS 형식론은 이산 사건 시스템을 객체 단위로 모듈

화 하여 계층적으로 결합하여 표현하는 집합론에 근거한 수학적인 틀이다. DEVS 형식론에는 시스템 기본적인 구성 요소를 나타내는 원자 모델과 여러 모델을 합쳐서 새로운 모델을 구성할 수 있는 결합 모델[1]이 있다 [4]. 원자 모델은 DEVS 형식론을 구성하는 가장 기본적인 모듈로서 시스템의 행동을 기술하는 모델이다. 원자 모델 M의 수학적인 표현은 다음과 같다.

$M = \langle X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta \rangle$
X: 이산사건 입력 집합
Y: 이산사건 출력 집합
S: 일련의 이산사건 상태의 집합
δ_{ext}: $Q \times X \rightarrow S$:외부 상태 천이 함수
$Q = \{ (s,e) \mid s \in S, 0 \le e \le ta(s) \}$: 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

원자 모델은 3개의 집합과 4개의 함수로 구현 되어 있다. 기존의 DEVS 실행 환경은 3개의 집합을 관리하면서 4개의 함수(δ_{ext}, δ_{int}, λ, ta)를 필요에 따라 호출시켜 모델을 실행한다. DEVS 모델을 실행하는 계층적 스케줄링 알고리즘은 [4]에 명시되어 있다.

계층적 스케줄링 알고리즘의 가장 큰 부하는 메시지 패싱과 모델 실행 시간 관리에서 발생한다[2]. 메시지 패싱은 한 모델에서 출력이 발생하여 다른 모델에게 입력 이벤트를 발생 시킬 때 일어나며 직접 전달되지 않고 계층적 구조를 따라 전달되기 때문에 비효율적이다. 모델 실행 시간 관리를 할 때는 매번 모든 모델에게 다음 실행 시간을 요청하고 그 중 최소 시간의 모델을 실행한다. 매번 모델을 실행 할 때 마다 O(n)의 시간 복잡도를 가지는 과정을 반복하기 때문에 비효율적이다. 시뮬레이션 속도 향상을 위해서 이 두 가지 문제를 다루어야 한다.

3. Event-oriented DEVS 모델 실행 환경

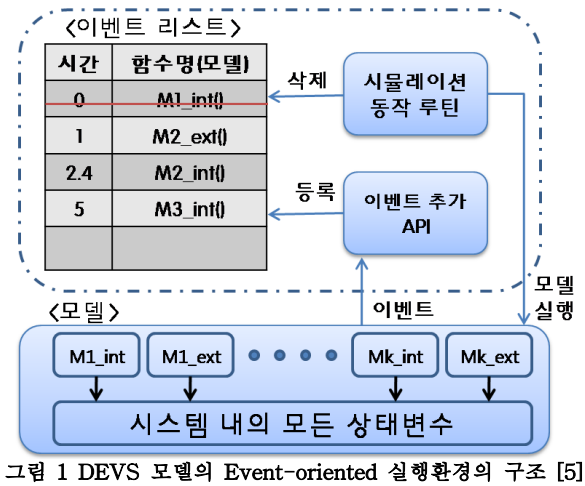

그림 1 DEVS 모델의 Event-oriented 실행환경의 구조 [5]

1) 제안하는 Event-oriented 기반의 실행 환경은 계층이 없기 때문에 본 논문에서는 결합 모델에 대해서 다루지 않는다.

Event-oriented 실행 환경은 실행 대상이 되는 모델의 함수와 이벤트 리스트, 상태 변수들로 구성이 되며 각 이벤트는 실행 시간과 실행할 함수에 대한 정보를 가진다.

그림 1은 전체 구조를 도식화한 그림이다. 모델 실행 환경은 이벤트 리스트와 시뮬레이션 동작 루틴, 이벤트 추가 API로 구성된다. 동작 루틴이 이벤트 리스트의 최상위 이벤트를 삭제하고 그 이벤트에 적힌 정보대로 함수를 실행한다. 상태변수들은 전역변수로 공유되는데, 각 함수는 필요에 따라 상태변수를 참조, 수정할 수 있다. 다음 실행될 함수를 지정할 땐, 이벤트 추가 API를 통하여 이벤트 리스트에 추가한다.

3.1 Event-oriented 실행 환경을 위한 DEVS 모델

이산 시스템의 상태 천이는 입력이 있을 때 일어나는 외부 천이와 입력이 없을 때 일어나는 내부 천이로 구분한다. DEVS의 함수를 위와 같은 관점에서 크게 두 개의 흐름으로 정리할 수 있다. 입력이 들어오면 δ_{ext}함수가 실행되고 다음 δ_{int}함수가 발생할 시간을 정의하는 ta 함수가 실행된다. 정해진 시간 동안 입력이 없게 되면 우선 λ함수가 실행되어 출력이 발생하고 δ_{int}함수가 실행 된 후, 다음 δ_{int}함수가 발생할 시간을 정의하는 ta함수가 실행된다.

위 내용에 기초하여 이벤트와 이벤트 리스트를 이용하여 실행하는 Event-oriented 실행 환경을 위한 DEVS 모델을 구현한다. 각 모델은 DEVS 형식론과 같은 표현력을 가지도록 설계되어 있으며 외부 천이 함수(external transition function)와 내부 천이 함수(internal transition function)로 구성되어 있다.

DEVS는 입력 X가 입력되거나 ta함수가 정한 시간이 다 지나면 모델이 실행되지만 Event-oriented 실행 환경에서는 이벤트에 의해 상태 천이 함수가 실행된다. 또한 이벤트가 등록되어야 다음 함수가 실행될 수 있으므로 각 함수는 필요에 따라 이벤트를 등록한다.

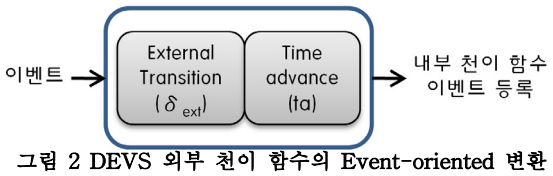

그림 2 DEVS 외부 천이 함수의 Event-oriented 변환

그림 2를 보면, 시뮬레이션 동작 루틴이 이벤트에 의해서 함수를 실행한다. 외부 천이 함수 내부에서는 DEVS의 δ_{ext}에 해당하는 코드를 실행하고 내부 천이 함수에 대한 이벤트가 등록된다. 다만 이때 다음 내부 천이 함수의 실행 시간을 얻기 위해 ta에 해당하는 코드를 실행해서 이벤트의 실행 시간을 알려준다.

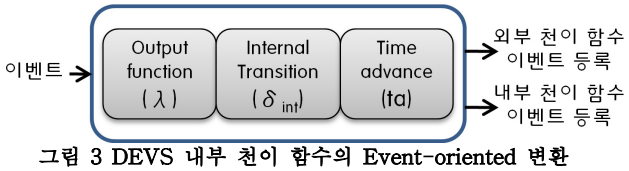

그림 3 DEVS 내부 천이 함수의 Event-oriented 변환

내부 천이 함수 역시 이벤트에 의해서 실행된다. 그림 3을 보면 실행된 함수는 출력을 발생시키는 λ함수와 δ_{int} 함수에 해당하는 코드를 실행한다. 외부 천이 함수와 마찬가지로 ta를 실행시키고 다음 이벤트의 실행 시간을 받아 다음 이벤트를 등록한다. 그림을 보면 함수를 순서대로 실행해야하는 것처럼 보일 수 있으나 반드시 그렇게 구현할 필요는 없으며 사용자가 자유롭게 구현할 수 있다.

그림 2에는 DEVS에서의 입력 X와 상태 변수 S에 대한 언급이 없는데 Event-oriented 실행 환경의 구조에서 언급한 듯이 모든 변수와 입력은 전역 변수로 선언되어 있고 모든 모델이 자유롭게 접근할 수 있다. 내부 천이 함수에서 출력이 발생하게 되면 해당 외부 천이 함수에 대한 이벤트를 등록하고(그림 3) 실제 출력 내용은 전역 변수에 기록 한다. 이것은 이벤트에 의해 호출된 외부 천이 함수에서 읽혀진다. 즉, 전역 변수를 이용하여 직접 전달하는 방식이다. 따라서 제안하는 실행 환경은 앞에서 말한 메시지 패싱으로 인한 부하가 존재하지 않는다.

4. Event-oriented 모델 실행 환경 설계 및 구현

4.1 실행 환경의 설계 고려사항

실행 환경을 위해서 다음 3가지 API가 구현되어야 한다. 시뮬레이션 동작은 한번 실행하면 시뮬레이션 종료까지 계속 되며 이벤트 등록 및 삭제는 모델에 의해서 계속해서 발생된다.

4.1.1 시뮬레이션 동작

시뮬레이션 시작 함수를 호출하면 시뮬레이션이 시작되고 이벤트 리스트가 빌 때 까지 동작한다. 이벤트 리스트로부터 이벤트를 뽑아내고 이벤트가 지시하는 함수를 실행한다. 이벤트에는 실행 시각이 기록되어 있고 실행과 함께 시스템의 현재 시간이 수정된다.

4.1.2 이벤트 등록

이벤트를 등록하는 API는 모델 어디서나 호출 될 수 있다. 모델에서는 API를 호출하면서 생성되어야 할 이벤트의 정보를 넘겨주면 필요한 이벤트를 만들어서 이벤트 리스트에 삽입한다. 이벤트 정보에는 실행할 함수와 실행 시간이 포함되어 있다.

4.1.3 이벤트 삭제

DEVS의 계층적 스케줄링 알고리즘은 각 모델마다 다음 실행 시간을 매번 업데이트해서 하나의 값으로 가지고 있다. 따라서 다음 실행 시간이 변하더라도 업데이트가 바로 이루어진다. 하지만 이벤트 리스트로 관리하게 되면 업데이트가 될 때 이전에 등록한 이벤트를 삭제해야 한다. 함수 이름과 함께 이벤트 삭제 API를 호출하면 이벤트 리스트에서 해당 이벤트를 찾아내어 삭제한다.

4.2 이벤트 리스트 구현

Event-oriented 모델을 실행 하는데 있어서 가장 큰 문제는 이벤트 리스트를 관리하는 것이다. 이벤트 리스트 구현 방법에 따라 앞의 API의 성능이 달라지기 때문에 이벤트 리스트를 구현하는 방법이 성능의 중요한 요소가 된다. 본 논문에서는 정렬된 리스트와 힙을 비교, 분석하였다.

표 1 시간 복잡도 비교

기능	정렬된 리스트	힙
최소 이벤트 삭제	O(1)	O(lg n)
이벤트 삽입	O(n)	O(lg n)
임의의 이벤트 삭제	O(n)	O(n)

표1은 실행 환경 구현에 있어서 필요한 작업들에 대해 시간 복잡도를 비교한 것이다. 이벤트 삭제가 거의 일어나지 않는다고 가정하면 평균적으로 O(lg n)의 시간 복잡도를 가지는 힙이 효율적이다. 정렬된 리스트는 최소의 이벤트를 가져오는 것에는 효율적이나 삽입을 할 때 각 이벤트를 비교하면서 순서를 결정해야 하므로 비효율적이다. 하지만 힙은 상수 값이 크다는 것을 고려해야한다. 즉, 리스트 길이에 따른 시간 증가는 비교적 완만하지만 상수 k가 커서 n이 작을 때는 오히려 리스트보다 더 나쁜 성능을 보일 수 있다.

위의 내용을 통해 n이 작을 때는 정렬된 리스트가 더 좋은 성능을 보이지만 n이 커지면서 힙이 더 좋은 성능을 보일 것이라는 예측을 할 수 있다. 5.2장에서 이 예측을 실험 결과로 확인할 것이다.

5. 실험 결과

제안한 모델과 환경의 성능을 시험하기 위해서 단일 서버 큐잉 모델을 사용하였다. 먼저, 이벤트 리스트의 구현에 따라서 어떤 성능을 얻을 수 있는 지를 시험하고 이어서 계층적 스케줄링 알고리즘과 성능을 비교한다. 계층적 스케줄링 알고리즘을 사용한 실행환경으로 KAIST SMSLab에서 개발한 DEVSim++를 사용하였다.

DEVSim++는 DEVS 형식론을 C++ 언어로 구현한 DEVS 모델 실행 환경이다 [6].

5.1 단일 서버 큐잉 모델

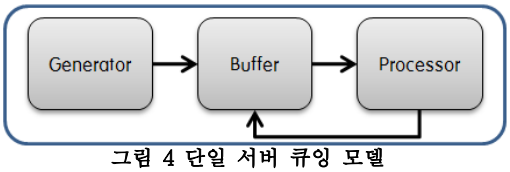

그림 4 단일 서버 큐잉 모델

그림 4는 단일 서버 큐잉 모델을 간략히 나타낸 것이다. 신호발생기(Generator)에서 일정한 주기로 신호를 발생시키면 버퍼(Buffer)는 신호를 저장하고 프로세서(Processor)로 신호를 보내준다. 프로세서는 바쁨과 대기, 두 가지 상태로 이루어져 있는데 신호를 처리하는 중에는

바쁨 상태이고 버퍼가 대기한다. 신호 처리가 끝나면 버퍼에게 끝났음을 알리고 대기 상태가 된다.

본 논문의 실험을 위해서 위의 모델을 DEVS로 구현하였다. n개의 큐잉 모델이 병렬적으로 수행되어 모델 개수와 이벤트 개수에 따른 실행 환경의 부하를 측정하였다.

5.2 이벤트 리스트에 따른 성능 비교 및 분석

정렬된 리스트와 힙으로 두 개의 실행 환경을 만들어서 실험했다. 한 번에 실행되는 큐잉 모델의 개수를 1부터 200까지 증가시켰다. 각 모델은 각각 10000개씩의 신호를 처리한다.

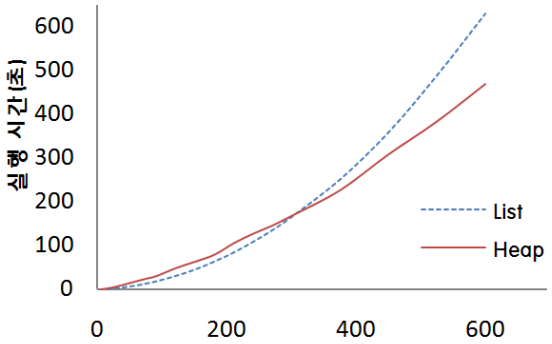

그림 5 이벤트 리스트 구현방식에 따른 비교

그림 5는 실험 결과를 이벤트 개수에 따라 나타낸 것이다. 이벤트의 개수가 300개 이하일 때는 정렬된 리스트가 빠르지만 300개 이상에서는 정렬된 리스트의 실행 시간이 급속히 증가한다. 힙은 시간 복잡도에서 우수해서 이벤트가 많아짐에 따라 유리하지만 시간 복잡도에서는 생략되는 상수 k값이 크다. 초반에는 상수 k값이 작은 정렬된 리스트가 더 빠르지만 점차 이벤트의 개수가 늘어나면서 힙이 더 좋은 성능을 보인다.

5.3 계층적 스케줄링 알고리즘과 성능 비교 및 분석

이벤트 리스트를 구현한 실행환경과 계층적 알고리즘을 사용한 실행 환경의 성능을 비교했다. 큐잉 모델의 개수를 1부터 500까지 증가시켰다.

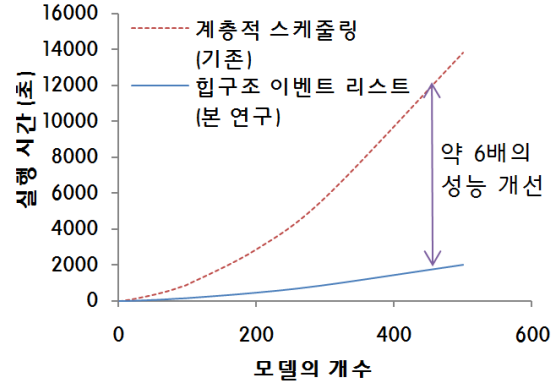

그림 6 DEVS 모델의 시뮬레이션 시간

그림 6은 실험 결과를 시뮬레이션 시간으로 나타낸 그래프이다. 제안한 실행환경이 더 좋은 성능을 나타내는 것을 볼 수 있다. 모델의 개수가 적어서 이벤트의 개수가 적을 때는 제안한 모델 실행 환경이 14배까지 빠르지만 점차 이벤트 개수가 늘어남에 따라 줄어들고 6배정도의 성능 향상을 확인 할 수 있다.

성능 향상의 원인은 이벤트 리스트를 통한 모델 실행이 2장에서 말한 계층적 스케줄링의 2가지 문제점을 해결하기 때문이다. 이벤트 리스트를 사용한 실행 환경에서는 입출력이 전역 변수를 통해 이루어지기 때문에 메시지 패싱에 대한 부하가 없고 다음 실행 시간이 이벤트 실행과 함께 업데이트되기 때문에 모델 실행 시간 관리가 O(1)에 이루어진다.

6. 결론

본 논문은 관점적으로 DEVS 형식론을 접근하여 DEVS 형식론을 실행하면서도 성능이 좋은 Event-oriented 실행 환경을 제안했다. 이벤트와 이벤트 리스트를 이용하여 모델을 실행하는 모델 실행 환경의 리스트를 힙으로 구현하여 이벤트 리스트로 인한 부하를 최소화 하였다.

DEVS 모델을 실행하는 Event-oriented 실행 환경은 이벤트 리스트를 이용하여 모델을 실행하는 알고리즘을 가지고 있기 때문에 계층적 스케줄링 알고리즘과 비교하여 부하를 줄였으며, 실험을 통해서 성능이 더 뛰어나다는 것을 확인했다. 모델의 수가 많아짐에 따라 이벤트 리스트 관리의 부하가 커지더라도 약 6배 정도 빠른 것으로 실험 결과를 얻을 수 있었다.

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] 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.

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

[5] Tag Gon KIM, IE801 Lecture Note, EE, KAIST, 2009, http://smslab.kaist.ac.kr/.

[6] 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.