이산사건 시스템을 위한 객체지향 모델링 및 시뮬레이션 환경

° 안 명수*, 박 성봉**, 김 탁곤*
* 한국과학기술원 전기 및 전자공학과, ** 삼성중공업 중앙연구소

An Object-oriented Modeling and Simulation Environment for Discrete Event Systems

Myung Soo Ahn*, Sung Bong Park**, Tag Gon Kim*
* Dept. of Electrical Eng., KAIST, ** Daeduk R&D Center, SHI

Abstract

This paper presents an object-oriented environment DEVSIM++ which supports modeling and simulation of discrete event systems. It realizes Zeigler's DEVS formalism in C++ as a semantic-based modeling formalism and the abstract simulator of the formalism for simulation. Developed all in C++, the DEVSIM++ environment fully exploits the advantages from the object-oriented paradigm and sound semantics from the DEVS formalism. Thus, DEVSIM++ supports the development of discrete event models within the DEVS framework and efficient simulation of the models.

1. 서론

최근들어 다중프로세서 컴퓨터, 광역 통신망, 교통제어 시스템등 복잡한 시스템의 설계 및 관리에서 컴퓨터를 이용한 성능평가의 중요성이 강조되고 있다. 시스템이론적으로 이러한 시스템들을 이산사건 시스템으로 분류하는데 현재까지 이와같은 시스템들을 잘 모델링하여 평가할 수 있는 해석적인 방법이 개발된 사례가 없기때문에 시뮬레이션에 의존할 수 밖에 없다. 특히 실시간 시스템등과 같이 시스템 설계시에 각 동작이 일어나는 시간이 중요한 경우에는 시뮬레이션을 사용하는 것이 가장 효율적이다[3].

이산사건 시스템의 정량적 특성을 측정하기 위하여 GASP, GPSS, SIMAN, SIMSCRIPT, SLAM등과 같은 많은 시뮬레이션 전용 언어들이 발표되었다[2]. 이들 언어들은 모델의 표현력과 시스템을 계층적으로 분해하여 기술할 수 있는 기능이 제한되므로 복잡한 시스템의 모델링 작업에 많은 비용이 요구된다. 위와같은 시뮬레이션 언어를 사용한 환경들의 단점을 보완하기 위하여 정형화된 형식론의 의미론에 기반한 새로운 이산사건 모델링 및 시뮬레이션 환경이 필요하다.

본 논문에서는 의미론에 기반하여 객체지향 언어인 C++로 구현된 새로운 이산사건 시스템 모델링 및 시뮬레이션 환경인 DEVSIM++에 대해 소개한다. DEVSIM++에서 모델링 형식론으로 채택한 Zeigler의 DEVS(Discrete EVent System Specification) 형식론은 집합론을 사용하여 이산사건 시스템을 시스템이론적으로 기술한다[7]. DEVS 형식론은 추상화된 시뮬레이터 알고리즘을 지원함으로써 새로운 시뮬레이션 언어/환경을 좀 더 명료한 의미론을 가지고 개발하는 것을 가능하게 한다.

DEVSIM++를 객체지향 환경으로 구현함으로써 DEVS 형식론으로 기술된 시스템 모델들의 재사용성 뿐만아니라 모델의 관리 및 수정을 용이하게 하였다. 본 논문에서는 DEVS 형식론과 이를 구현하는 DEVSIM++ 환경에 대해 기술한 후에 간단한 시스템의 모델링 및 시뮬레이션 방법에 대해 설명한다.

2. DEVS 형식론

이산사건 시스템은 서로 상호 통신하며 동작하는 요소들의 집합으로 관찰될 수 있는데 이러한 관점에서 이산사건 모델링을 위해서는 모델링 대상인 객체들을 명료하고 일반적인 방법으로 표현하고 이들 사이의 통신을 일관성있는 형식으로 기술할 수 있는 기능이 필요하다. 이점을 고려하여 이산사건 시스템을 시스템이론적으로 표현하는 형식론이 DEVSIM++에서 채택한 Zeigler의 DEVS 형식론이다. 본 절에서는 DEVS 형식론과 이를 사용하여 모델링된 모델들을 시뮬레이션 하는 알고리즘에 대해 간략하게 기술한다.

2.1 DEVS 형식론

Zeigler에 의해 제안된 DEVS 형식론은 이산사건 시스템을 모듈별로 나누어서 계층적으로 모델링할 수 있는 방법을 제공한다. DEVS 형식론은 집합이론에 기반을 두고 있으며 시스템을 계층적으로 분해해서 기술한 다음 이들을 결합할 수 있도록 하기 위하여 위하여 두가지 모델 클래스, 즉, atomic DEVS 모델과 coupled DEVS 모델로 나누어서 모델을 표현하고 있다. 시스템의 동작은 각 atomic 모델들이 서로 사건(event)을 주고 받으며 시간에 따라 상태를 변화하는 과정으로 표현되며 coupled 모델은 각 구성요소 모델간의 사건 전달 경로를 정한다.

Atomic 모델은 더 이상 나눌 수 없는 모델로서 시스템의 구성 요소의 동적특성을 시간명세 상태전이시스템(timed state transition system) 레벨에서 기술 한다. Atomic 모델 M 은 다음과 같은 항들로 명세할 수 있다 :

$$M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta > .$$

각 항들의 의미는 다음과 같다.

X : 외부입력 사건 집합;
S : 상태 집합;
Y : 출력사건 집합;
$\delta_{int} : Q \times X \rightarrow S$: 외부변이 함수;
$\delta_{ext} : S \rightarrow S$: 내부변이 함수;
$Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$: total state;
$\lambda : S \rightarrow Y$: 출력 함수;
ta : $S \rightarrow Real$: 시간전진 함수.

상태변수 집합은 모델이 가질 수 있는 모든 가능한 상태를 포함한다. 모델의 상태는 외부에서 입력 사건을 받았을 때나 현 상태에서 내부적으로 정해진 시간이 경과했을 때 변화되는데 각각 외부변이 함수와 내부변이 함수에 의해 상태전이 규칙이 정해진다. 시간전진 함수란 외부 입력 사건없이 한 상태에서 머물 수 있는 최대한의 시간을 정하며 이 시간이 경과하면 내부전이를 겪게된다. 출력 함수에서는 각 상태에서 어떠한 출력 사건을 발생시키는가를 정한다.

계층적으로 분해되어 모델링된 시스템의 구성요소 모델들을 서로 결합하는 기능은 두번째 모델 형식인 coupled 모델에 의해서 제공된다. Coupled 모델은 시스템의 각 구성요소 모델들이 서로 어떻게 연결되어 신호를 교환하는 지를 기술한다. Coupled 모델은 atomic 모델를 또는 coupled 모델들을 그의 구성요소로 가지며 각 구성요소들과 요소들간의 연결상태를 다음과 같은 항들을 사용하여 표현한다 :

$$DN = < D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, SELECT > .$$

각 항들은 다음과 같은 의미를 갖는다.

D : 구성요소들의 이름 집합;
D의 각 구성요소 i에 대해

M_i : 구성요소 i의 DEVS 모델;
I_i : i의 influencees;
$Z_{i,j}$: $Y_i \rightarrow X_j$: i로부터 j로의 출력변환 함수;
$SELECT$: $D \rightarrow D$의 subsets : tie-breaking selector.

구성요소 i의 influencees란 i의 출력단자에 연결된 요소들을 말한다. 집합 I_i의 각 요소들을 j로 나타내었을 때 $Z_{i,j}$는 i에서 j로의 출력 변환을 나타내는 함수이다. 즉, i의 어떤 출력단자가 j의 어느 입력단자로 연결되는 지를 기술한다. $SELECT$로 나타낸 항은 두개 이상의 구성요소들의 다음 내부 변이 시간이 같을 때 어느 요소부터 내부 변이를 시킬 것인지 결정하는 함수이다.

DEVS 형식론에 대한 자세한 설명은 [7]를 참조하면 된다.

2.2. 추상화된 시뮬레이터

DEVS 형식론에서는 명세된 모델의 동적인 특성을 해석하여 시뮬레이션하는 알고리즘으로 추상화된 시뮬레이터를 제공한다. 추상화된 시뮬레이터는 두가지 클래스로 이루어져 있는데 하나는 atomic 모델을 위한 *simulator*이고 다른 하나는 coupled 모델을 위한 *coordinator*이다. 각 atomic 모델에는 해당되는 simulator가 결합되어 모델의 실행을 제어한다. Coordinator는 coupled 모델의 구성요소 모델사이의 실행 순서를 제어하며 각 구성모델사이의 통신기능을 도와준다.

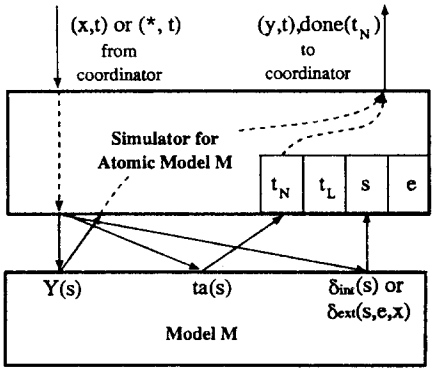


그림 1: Abstract Simulator for Atomic DEVS Model M.

이산사건 시스템의 동작을 시스템의 상태전이로 해석하고 atomic 모델의 상태전이는 외부사건과 내부사건에 의해서만 발생하므로 simulator는 해당되는 atomic 모델에 전달되는 외부변이와 내부변이 신호를 전달받아 atomic 모델이 정해진 상태전이를 하도록 제어한다. 그리고 모델로부터 다음 내부변이 시간을 전달받아 정해진 시간에 모델이 스케줄될 수 있도록 모델이 발생한 출력신호를 상위 시뮬레이터에게 전달한다. 그림 1는 simulator의 동작을 보여준다.

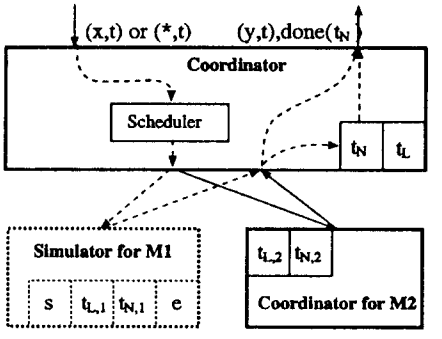


그림 2: Abstract Simulator for Coupled DEVS Model.

Coupled 모델을 위한 coordinator는 각 구성요소 모델들간의 통신, 즉, 메시지 전달 기능을 수행하며 다른 모델로부터 전달되는 외부사건 신호를 해당되는 모델의 simulator 또는 coordinator에게 전달한다. 모델들의 내부변이 시간을 관리하여 내부변이 신호가 들어오면 가장 먼저 수행될 모델에게 내부변이 신호를 전달한다. 만약, 두개이상의 모델이 동시에 수행되어야 하면 정해진 선택 규칙에 의해 한 모델을 선택하여 스케줄한 후에 다음 모델을 스케줄한다. 그림 2는 모델 M1과 M2로 구성되는 coupled 모델을 위한 coordinator의 동작을 보여준다. 그림에서는 모델 M2가 스케줄링된 상태이다.

위와같이 DEVS 형식론으로 모델링된 시스템을 실시간으로 시뮬레이션할 수 있는 잘 정의된 알고리즘이 지원되므로 모델링 및 시뮬레이션을 단일화된 환경하에서 지원할 수 있는 환경구축이 용이하다.

3. DEVSIM++ : DEVS 형식론의 객체지향 구현

DEVSIM++은 Zeigler에 의해 제안된 DEVS 형식론을 시스템 모델링 도구로 채택하였으며 DEVS 형식론의 추상화된 시뮬레이터를 시뮬레이션 엔진으로 사용한 이산사건 시스템 모델링 및 시뮬레이션 환경이다. 객체 지향 언어인 C++로 모든 환경을 구현함으로써 DEVS 형식론의 모듈화, 계층적 시스템 모델링 의미론을 완전히 소화하였으며 모델 표현력과 효율성을 높일 수 있었다.

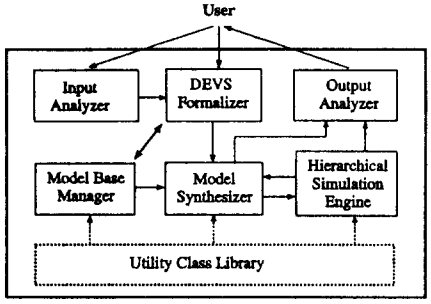


그림 3: System Architecture of DEVSIM++.

그림 3는 DEVSIM++ 구조를 보여주고 있다. DEVS formalizer는 사용자에게 시스템 모델링 기능을 제공하며 모델링된 모델들을 모델베이스에 관리하여 재사용할 수 있게 한다. 각 모델에는 해당되는 시뮬레이터가 결합되어 모델의 실행을 제어하는데 엔진은 2절에서 설명한 추상화된 시뮬레이터를 객체지향으로 구현한 것이다. 입력데이타 분석기는 실제 시스템등으로부터 얻은 데이타의 특성을 분석하여 시뮬레이션에 사용될 확률함수로 변환하는 기능을 제공한다. 출력데이타 분석기를 사용하여 시뮬레이션 결과로 생성되는 각종 데이타의 통계적 분석을 수행할 수 있다.

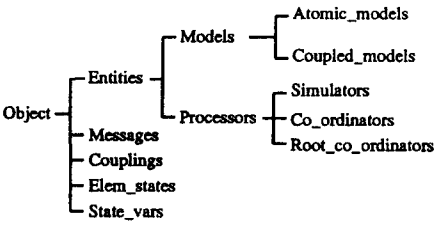
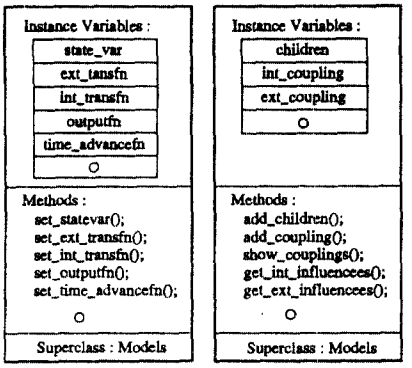


그림 4: Class Hierarchy of DEVSIM++.

DEVSIM++는 그림 4과 같은 클래스 계층 구조를 갖도록 객체지향 개념을 사용하여 설계되었다. 클래스 계층 구조는 NIH 클래스 라이브러리[4]을 기반으로 하여 설계되었는데 *Object* 클래스는 파생된 클래스들에게 공통으로 필요한 변수들과 함수들을 정의하는 root 클래스이다.

Entities 클래스는 DEVS 형식론과 추상화된 시뮬레이터를 구현한 클래스들의 가장 상위 클래스로서 객체의 이름을 저장하기위한 name이라는 변수와 이 변수를 다루기위한 함수들을 선언하고 있다. 이로부터 모델링을 위한 *Models* 클래스와 시뮬레이션을 위한 *Processors* 클래스가 파생되어 모델링과 시뮬레이션 부분을 완전히 분리된다.

Models 클래스는 모델링을 위한 클래스들의 상위 클래스로서 atomic 모델을 위한 *Atomic_models*와 coupled 모델을 위한



(a) Class Atomic_models. (b) Class Coupled_models.

그림 5: Specialization of *Models* Class.

*Coupled_models*가 공통으로 가지고 있는 변수와 method들을 가지고 있다. 이를 위한 변수들로는 inports(입력 단자 list), outports(출력 단자 list), processor, priority등이 있다. 여기에서 processor는 모델과 연관되어 있는 시뮬레이터의 pointer를 가지고 있고 priority는 coupled 모델에서 select 함수를 수행하기 위해서 각 모델들이 자기 고유의 우선순위를 가지고 있기 위한 것이다. *Model* 클래스에는 입출력 단자들을 지정하고 확인하기 위한 함수들이 선언되어 있다.

Atomic_models 클래스는 DEVS 형식론중 atomic 모델을 구현한 것으로서 atomic 모델에서 시스템 기술에 필요한 입출력 단자와 상태 집합, 그리고 내외부 상태 전이 함수, 시간전진 함수와 출력함수를 지정하기 위한 변수들과 사용자가 이들을 지정할 수 있는 함수들을 제공한다. Coupled 모델을 구현하는 *Coupled_models* 클래스는 children, int_coupling, ext_coupling, 그리고priority들의 변수와 이들을 정의할 수 있는 멤버함수를 정의한다. 그림5는 *Models* 클래스로부터 파생된 *Atomic_models*과 *Coupled_models* 클래스 구조를 보여준다.

추상화된 시뮬레이터를 구현하기 위해 상위 클래스로 *Processors* 클래스를 정의하고 하위 클래스로서 simulator를 구현하기위한 *Simulators* 클래스와 coordinator를 구현하기 위한 *Co_ordinators* 클래스를 두었다. 이들은 3절에서 설명한 알고리즘을 그대로 구현하며 자기가 관리하고 있는 모델들에 대한 정보를 갖고 있으며 다른 simulator 또는 coordinator들과 메시지 전달을 위한 기능을 제공한다. 또한 *Root_co_ordinators* 클래스가 있는데 이는 전체 시뮬레이션의 시작과 끝등을 총괄한다.

Messages, *Couplings*, *Elem_states*, *State_vars* 클래스들은 모델링과 시뮬레이션을 위한 클래스에서 사용하기 위한 데이타구조들을 정의한다. *Messages* 클래스는 각 모델사이의 신호 전달을 위한 데이타구조를 정의하며 *Couplings* 클래스는 coupled 모델에서 구성모델 사이의 연결 명세를 위해서 필요하다. *Elem_states* 클래스와 *State_vars* 클래스는 모델의 상태 변수를 리스트로 저장하기 위한 클래스들이다.

4. DEVSIM++ 환경에서 시스템 모델링 및 시뮬레이션

DEVSIM++환경에서 이산사건 시스템을 모듈화하여 계층적으로 모델링하여 시뮬레이션하는 과정을 설명하기 위하여 본 절에서는 간단한 프로세싱 모듈을 예제로하여 설명한다.

4.1 시스템 구성요소 식별

시스템을 DEVS 형식론으로 모델링하기 위해서는 모듈화하여 계층적으로 나타내기 위해서는 우선 시스템의 기본적인 구성요소들을 식별하고 이들 구성요소들을 계층적으로 연결하여야 한다. 구성요소 식별은 시뮬레이션의 목적과 밀접하게 관련이 있으며 시스템의 특정 구성요소의 동작이 중요한 경우에는 그 요소를 자세하게 분해하여 모델링할 수 있다.

그림 6은 간단한 프로세싱 모듈 PEL을 모델링하기 위하여 모듈을 BUF와 PROC 요소로 분해하고 이들 요소사이의 연결을 보여주고 있다. BUF는 "in"단자를 통해 외부로부터 들어오는 처리요구를 queue에 저장하여 PROC이 준비되어 있으면 out"단자를 통해 PROC에게 전달하며 PROC은 이를 작업이 종료하여 다음 처리를 할 준비가 되어 있다는 신호를 "done"단자로 BUF에게 보낸다. 일정량 이상의 작업이 수행되면 PROC은 "out"단자를 통해 외부에 처리요구 중지를 알린다.

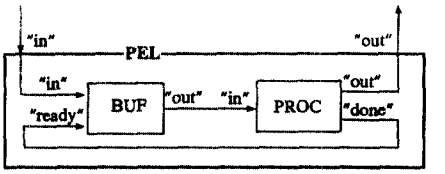
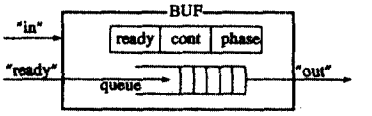
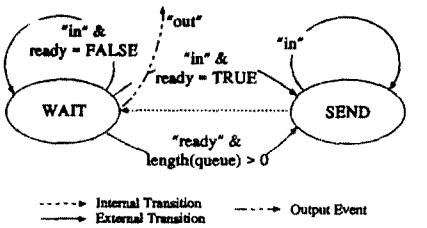


그림 6: Coupling Scheme of PEL.

시스템 구성요소 식별이 끝나면 다음 단계인 atomic 모델 개발을 용이하게 하기 위하여 각 구성요소의 구조와 동작을 분석하는 것이 좋다. Atomic DEVS 모델은 구성요소를 시간명세 상태전이 시스템으로 기술하므로 구성요소의 입출력 사건과 상태전이 도표를 그리는 것이 유용하다. 그림 7는 BUF 구성요소의 입출력 단자와 두 phase에서 동작하는 상태전이를 보여준다.



(a) I/O Ports and State Variables.



(b) Phase Transition Diagram of BUF.

그림 7: Atomic Model MBUF.

4.2. 모델 개발 방법

앞 절에서와 같이 각 구성요소의 동작에 대한 이해가 완료되면 이를 DEVSIM++에서 atomic 모델로 모델링할 수 있다. 먼저 atomic DEVS 모델의 4가지 특성함수인 내부/외부 전이함수, 출력함수, 시간전진함수를 구현한다. 다음 프로그램은 BUF 모델의 4가지 특성함수이다.

```
// external transition function //
void BUF_ext_transfn(State_vars& s, timeType& e,
                     Messages& message)
{
   OrderedCltn& qbuf = s.get_value("queue");
   if (message.get_port() == "in")
      qbuf.add(message.get_value());
      if (s.get_value("phase") == WAIT
            && s.get_value("ready") == TRUE)
         s.set_value("phase",SEND);
      else // continue
         s.set_value("cont",TRUE);
   } else if (message.get_port() == "ready")
      s.set_value("ready",TRUE);
      if (!qbuf.isEmpty())
         s.set_value("phase",SEND);
      else // continue
         s.set_value("cont",TRUE);
   }
}
```

```
// internal transition function //
void MBUF_int_transfn(State_vars& s)
{
   OrderedCltn& qbuf = s.get_value("queue");a
   if (s.get_value("phase") == SEND) {
      s.set_value("ready",FALSE);
      qbuf.remove();
      s.set_value("phase",WAIT);
   }
}

// output function //
void MBUF_outputfn(State_vars& s)
{
   extern Messages message;
   if (s.get_value("phase") == SEND)
      message.set("out",TRUE);
}

// time advance function //
timeType MBUF_time_advancefn(State_vars& s)
{
   if (s.get_value("cont") == TRUE) {
      s.set_value("cont",FALSE);
      return HOLD; // continue
   } else if (s.get_value("phase") == SEND)
      return 1
   else
      return INFINITY;
}
```

특성함수가 기술되면 이들을 *Atomic_models* 클래스로 부터 생성된 객체에 이를 지정해 주고 모델의 상태변수와 입출력단자들을 기술하면 된다. 다지막으로 모델의 초기상태를 정의하면 완전한 atomic 모델이 개발된다. 다음 프로그램은 위에서 구현한 특성함수들을 사용하여 BUF 모델을 개발하는 방법을 보여준다.

```
// initialize an object for atomic model BUF
void assign_BUF_functions(Atomic_models& buf)
{
   // define I/O ports and state variables//
   buf.add_inports(2,"in","ready");
   buf.add_outports(1,"out");
   buf.set_state_var(4,"cont","ready","phase","queue");

   // initialize the state variables//
   buf.set_state_value("phase",WAIT);
   buf.set_state_value("cont",FALSE);
   buf.set_state_value("ready",FALSE);
   buf.set_state_value("queue",new OrderedCltn());

   // set characteristic functions //
   buf.set_ext_transfn(BUF_ext_transfn);
   buf.set_int_transfn(BUF_int_transfn);
   buf.set_outputfn(BUF_outputfn);
   buf.set_time_advancefn(BUF_time_advancefn);
}
```

Atomic 모델들이 모두 개발되면 이들을 사용하여 coupled 모델을 개발한다. Coupled 모델의 개발은 모델의 구성요소 모델을 정의하고 이들 모델사이의 연결명세를 지정해주면 된다. 연결명세는 $(M1.p1, M2.p2)$와 digraph 형태로 지정된다. 이는 모델 $M1$의 p1단자로 발생되는 출력사건이 모델 $M2$의 $p2$ 입력단자를 통해 $M2$에 전달된다는 것을 의미한다. 다음은 그림 6의 PEL 모델을 개발하는 프로그램이다.

```
void create_pel(Coupled_models& pel)
{
   // define components and I/O ports
   pel.add_inports(1, "in");
   pel.add_outports(1, "out");
   pel.add_children(2, buf, proc);

   // internal couplings
   pel.add_coupling(buf, "out", proc, "in");
   pel.add_coupling(proc, "done", buf, "ready");

   // external couplings
   pel.add_coupling(&pel, "in", buf, "in");
   pel.add_coupling(proc, "out", &pel, "out");
}
```

4.3. 시뮬레이션 및 출력분석

모델링 작업이 끝나면 모델들을 객체화하여 실행하게 된다. 모델합성기에서 시스템 모델을 구성하기 위하여 필요한 모델들을 모델베이스에서 제공받아 이를 생성함으로써 이루어 진다. 모델합성기는 모델의 객체화시에 각 모델에 해당되는 시뮬레이터를 결합시켜 준다. 시뮬레이션은 *Root_co_ordinators* 클래스 객체를 생성하여 시스템 모델에 해당되는 객체를 지정한 후에 start라는 method를 실행하면 시작된다. 시뮬레이션은 모든 모델들이 수동상태에 도달하면 종료한다. 시뮬레이션 결과의 통계적 분석을 위하여 출력분석기가 제공된다. 출력분석기는 데이타로부터 평균, 분산등과 같은 점추정과 이를 확장한 confidence interval과 같은 구간추정등의 통계자료를 산출한다.

5. 결론

본 논문에서는 이산사건 시스템을 객체지향 관점으로 모델링하여 시뮬레이션할 수 있는 환경을 제공하기 위하여 개발된 DEVSIM++ 환경에 대하여 기술하였다. 또한, DEVSIM++ 환경하에서 모델개발 및 시뮬레이션 방법을 간단한 프로세싱 모듈을 예제로 하여 설명하였다.

DEVSIM++은 이산사건 시스템을 모듈화하여 계층적으로 기술하는 DEVS 형식론을 모델링 형식론으로 채용함으로써 모델 표현력의 제한요소를 해결하였다. 모든 환경을 객체지향 프로그래밍 언어인 C++로 구현하여 DEVS 형식론의 의미론을 충분히 이해하여 환경을 구축하였고 모델의 재사용성과 같은 객체지향 개념의 여러 장점들을 활용하였다. DEVSIM++을 사용하여 다중프로세서 시스템의 통신망 및 라우팅 알고리즘 설계과정에 실제 응용되고 있으며 실시간 시스템에서의 통신 방법에 관한 시뮬레이션 결과가 발표되었다[1].

현재 복잡한 이산사건 시스템 시뮬레이션시에 많은 모델들이 필요하게 되어 과도한 메모리를 요구하게 되는 단점을 해결하기 위해 실행시에 모델들을 화일에 관리하기 위한 방법에 대한 연구가 진행중이다. 아울러 완전한 실험 환경을 지원하기 위하여 입력 데이타 및 시뮬레이션 결과를 통계적으로 분석하는 기능에 대한 확장과 그래픽 사용자 환경을 지원하기 위한 기능도 추가되고 있다.

참고 문헌

[1] Myung S. Ahn and Tag G. Kim, "DEVS Methodology for Evaluating Time-Constrained Message Routing Policies", *Discrete Event Dynamic Systems : Theory and Applications*, 3, pp.173 - 192, 1993.

[2] C.G. Cassandras, *Discrete Event Systems : modeling and performance analysis*, Richard D. Irwin, Inc., and Aksen Associates, Inc., 1993.

[3] R.F. Garzia, M.R. Garzia, and B.P. Zeigler, "Discrete Event Simulation", *IEEE Spectrum*, Vol.23, No.12, PP.32-36, 1986.

[4] Keith E.Gorlen, Sanford M.Orlow and Perry S.Plexico, *Data Abstraction and Object-oriented Programming in C++*, John Wiley & Sons, New York, NY, 1990.

[5] Tag G. Kim, "Hierarchical Development of Model Classes in The DEVS-Scheme Simulation Environment," *Expert Systems with Applications*, Vol.3, No.3, pp.343-351, 1991.

[6] Sung B. Park, *DEVSIM++ : A Semantics Based Environment for Object-Oriented Modeling of Discrete Event Systems*, MS Thesis, KAIST, 1993.

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