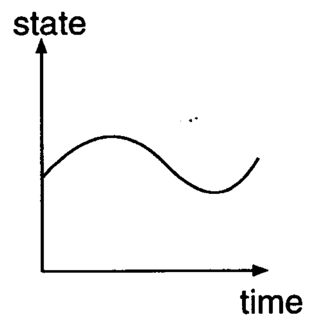
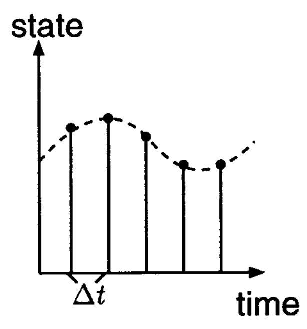
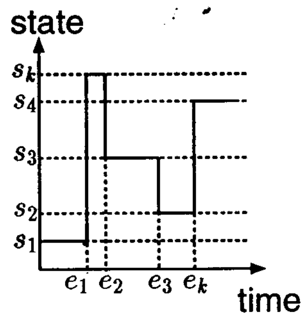
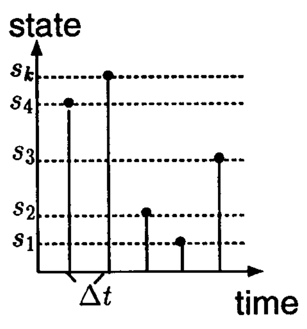
DEVSim++ : C++ Environment for Modeling/Simulation of DEVS Models

김 탁 곤

한국과학기술원
전기및전자공학과
컴퓨터공학연구실

1994. 6. 25

Systems Taxonomy

		TIME Space	
		continuous	discrete
STATE Space	continuous	• Continuous Systems • Differential Eqn. • Analog Circuits 	• Sampled Data Systems • Difference Eqn. • DSP 
	discrete	• Discrete Event Systems • DEVS Formalism • Distributed Systems 	• Digital Systems • Finite State Machine • Digital Circuits 

Examples of DES and Terminology Comparison

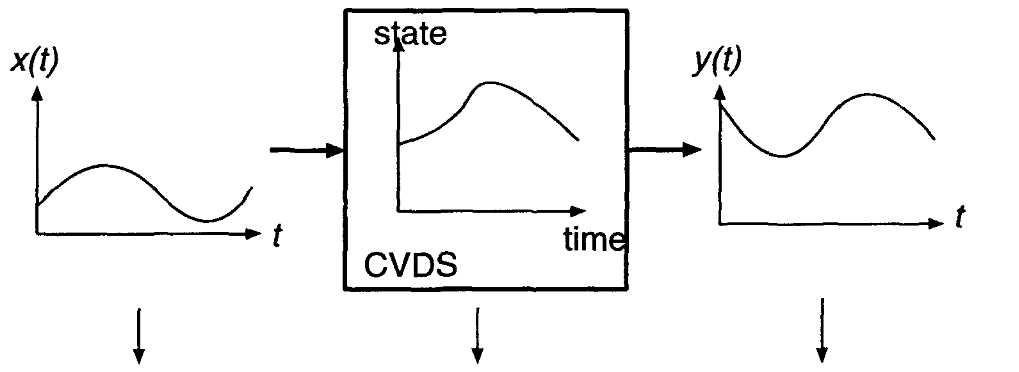
- *Multi-computer systems*
- *Communication networks*
- *Traffic systems*
- *Manufacturing systems*
- *War game*

$\longrightarrow$ *Man-made Systems*

Computer Communication System	Flexible Manufacturing System
Messages	Parts
Nodes	Work Stations
Virtual Circuit	Routes
Communication Links	Material Handling Systems
Packets	Automatic Guided Vehicles
Tokens	Fixtures

Comparison of CVDS and DEDS

CVDS : Time-driven Change of Continuous States

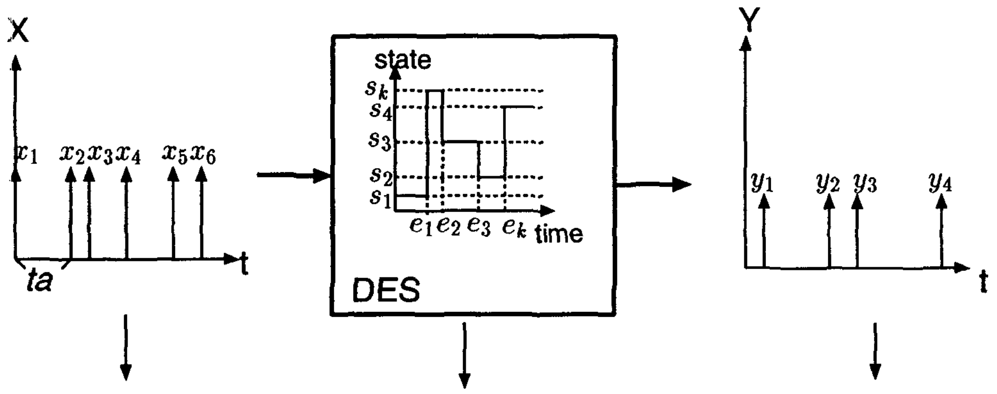


$x(t) = A \cdot sin(\omega t)$

$$\hat{Q} = AQ + BX$$
$$Y = CQ + DX$$

$y(t) = B \cdot sin(\omega t + \theta)$

DEDS : Event-driven Change of Discrete States

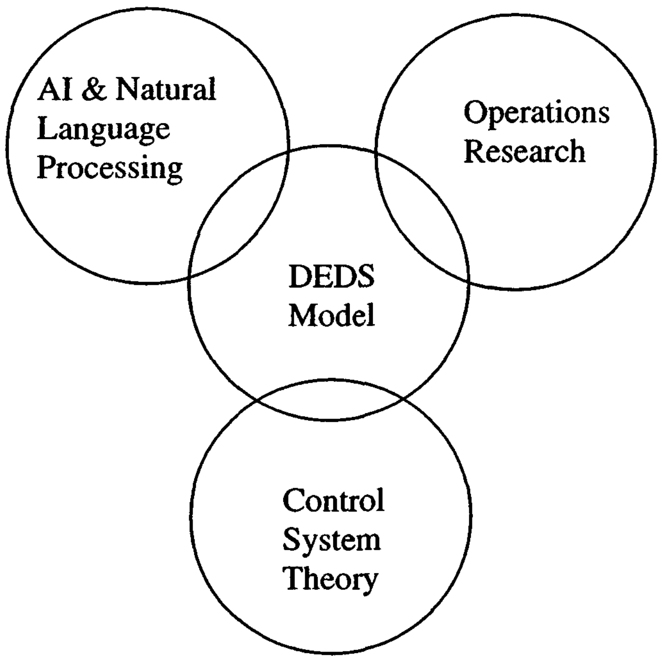

X
x_1 x_2 x_3 x_4 x_5 x_6
ta
t
state
s_k
s_4
s_3
s_2
s_1
e_1 e_2 e_3 e_k time
DES
Y
y_1 y_2 y_3 y_4
t

GENERATOR$_{DEVS}$ SYSTEM$_{DEVS}$ ACCEPTOR$_{DEVS}$

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

	CVDS	DEDS
Input	Continuous Time Function	Sequence of Events
State Trajectory	Continuous Time Function	Piecewise Constant Time Function
Output	Continuous Time Function	Sequence of Events

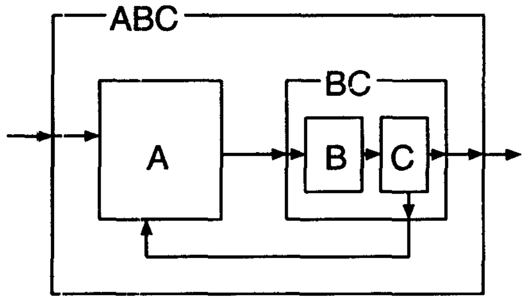
DES Model Boundary and Modelling Formalism

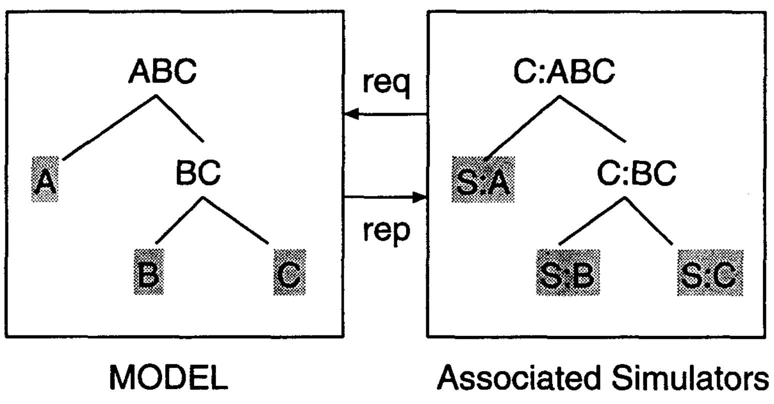


	Logic Base	Algebraic Base	Set/Bag theory Base	
Logical Analysis	Temporal Logic	Finitely Recursive Process(FRP); Communication Sequential Process (CSP);	Finite State Machines (FSM); Finite State Automata (FSA); Petri-Nets (PN)	Discrete Systems Specification (DEVS) Formalism
Performance Analysis		Min-Max Algebra	Timed PN	

Discrete Event Systems Specification (DEVS) Formalism

- Developed by Zeigler from mid 70's at U. of Michigan
- Set Theory Based Formal Specificaiton
- System Theoretic Representation of Discrete Event System
- Hierarchical, Modular Specification
- Associated Abstract Simulator
- Atomic Model and Coupled Model



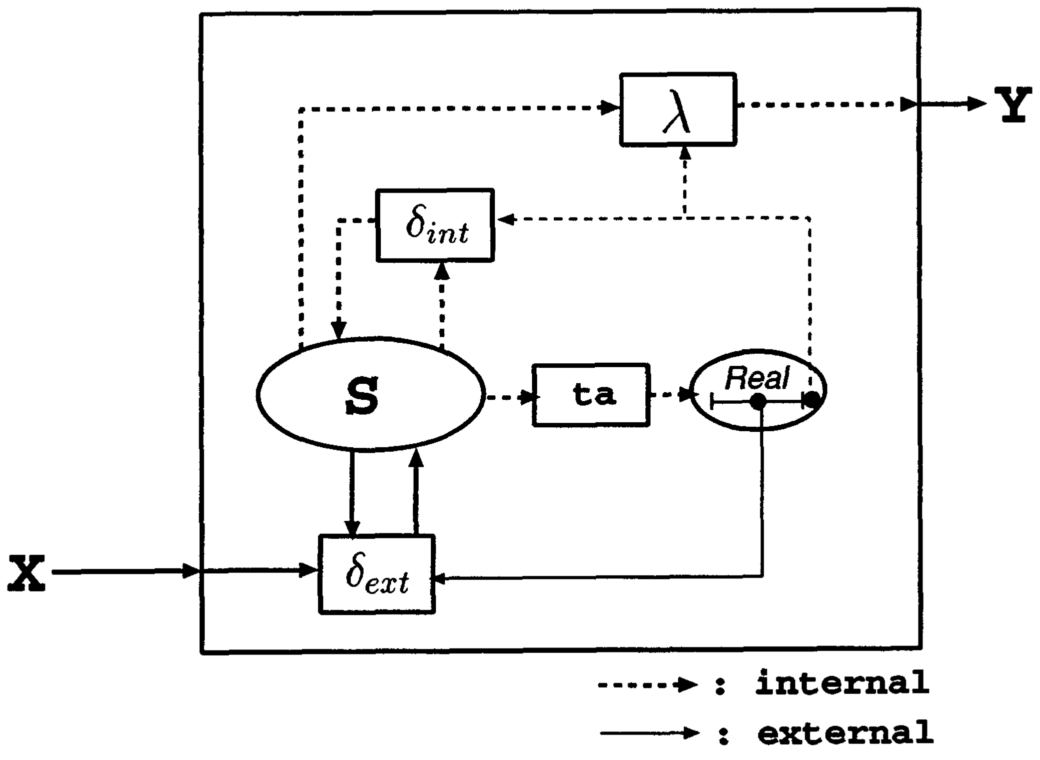


Atomic Model : Specification of Basic Model Behavior

Coupled Model: Specification of Hierarchical Model Structure

Abstract Simulator : Interpreter for Dynamics of DEVS Models

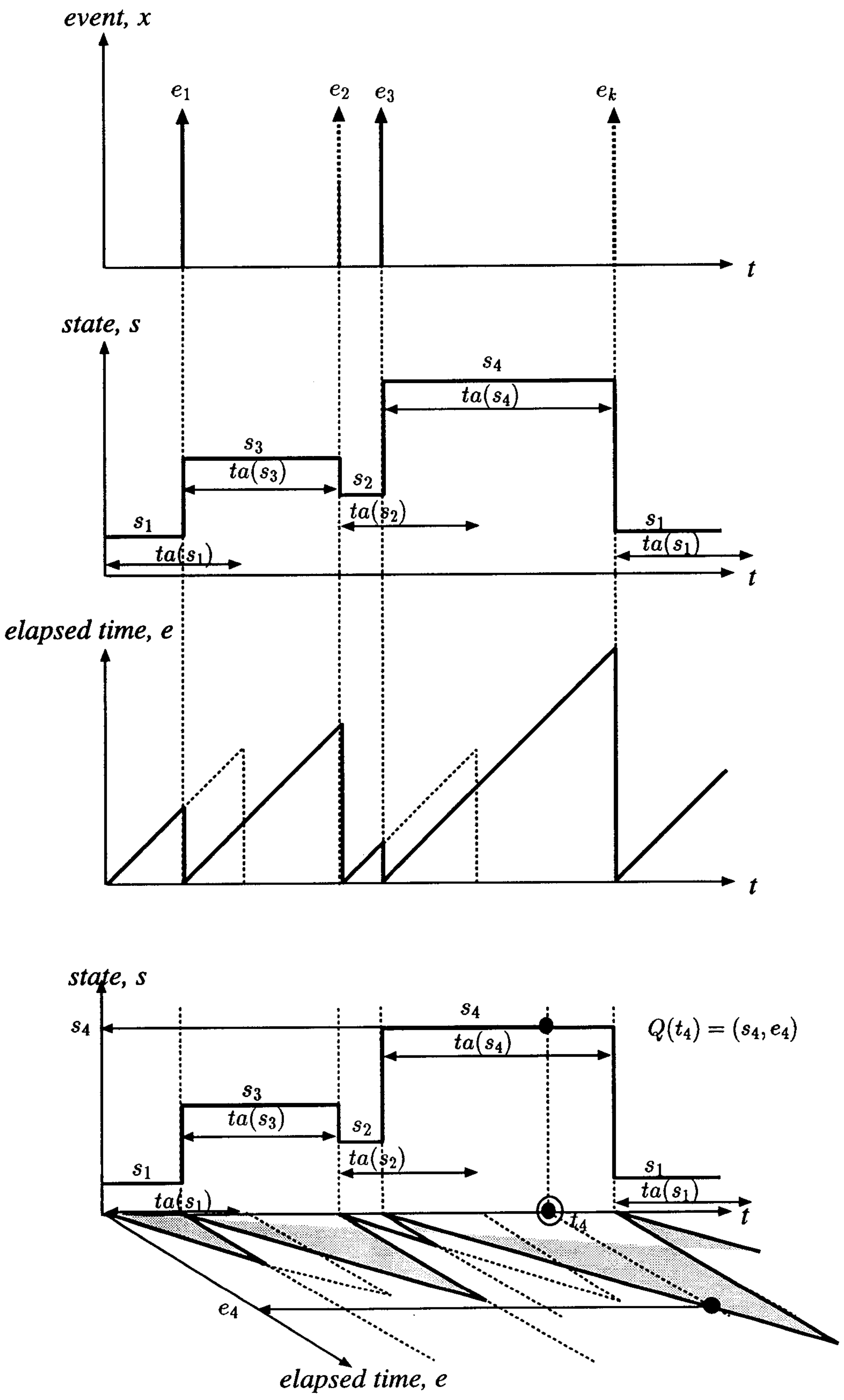
DEVS Specification for Atomic Model



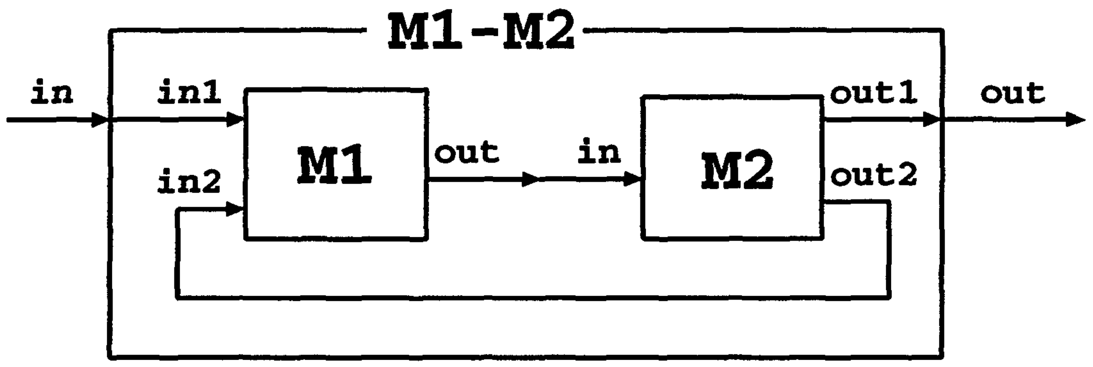
$$M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$$

X : input events set;
S : sequential states set;
Y : output events set;
δ_{int} : $S \rightarrow S$: internal transition function;
δ_{ext} : $Q \times X \rightarrow S$: external transition function;
$Q = \{(s,e) \mid s \in S, 0 \leq e \leq ta(s)\}$: total state of M;
$\lambda : S \rightarrow Y$: output function;
$ta : S \rightarrow Real$: time advance function.

Total State Q = (S, e)



DEVS Specification for Coupled Model



$$DN = < X, Y, M, EIC, EOC, IC, SELECT >$$

X : input events set;

Y : output events set;

M : set of all component models in DEVS;

$EIC \subseteq DN.IN \times M.IN$: external input coupling relation;

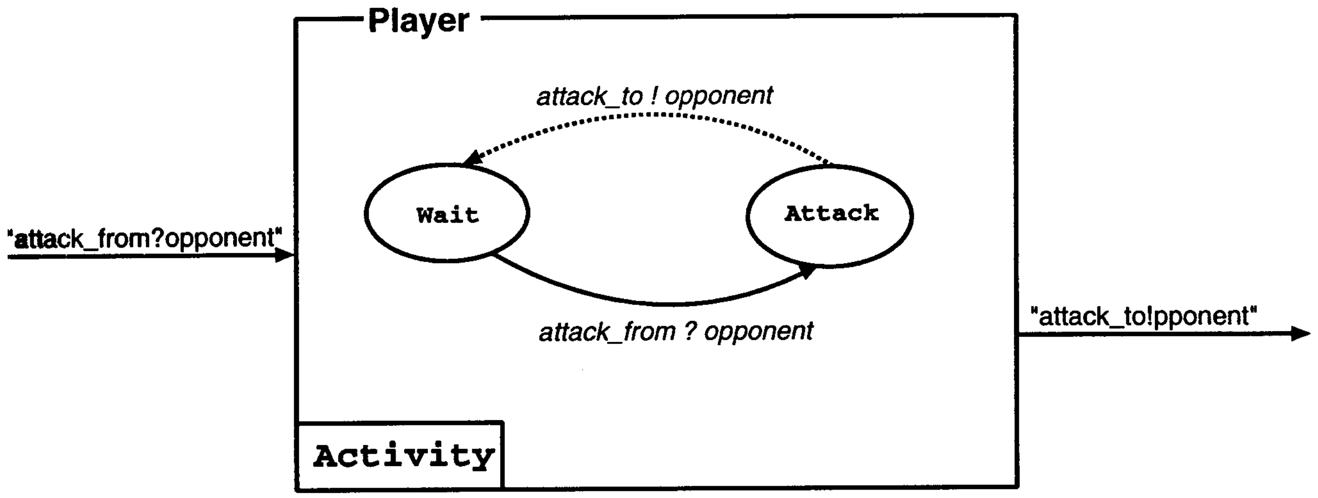
$EOC \subseteq M.OUT \times DN.OUT$: external output coupling relation;

$IC \subseteq M.OUT \times M.IN$: internal coupling relation;

$SELECT : 2^M - \emptyset \rightarrow M$: tie-breaking selector,

where the extention $.IN$ and $.OUT$ represent the input ports set and the output ports set of respective DEVS models.

An Example: Ping-Pong game Player



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

$X = \{ \text{ attack\_from?opponent } \}$

$Y = \{ \text{ attack\_to!opponent } \}$

$S = \{\text{activity} \mid \text{activity} \in \{\text{Wait, Attack}\} \}$

$\delta_{ext}(Wait, attack\_from!opponent) = Attack$

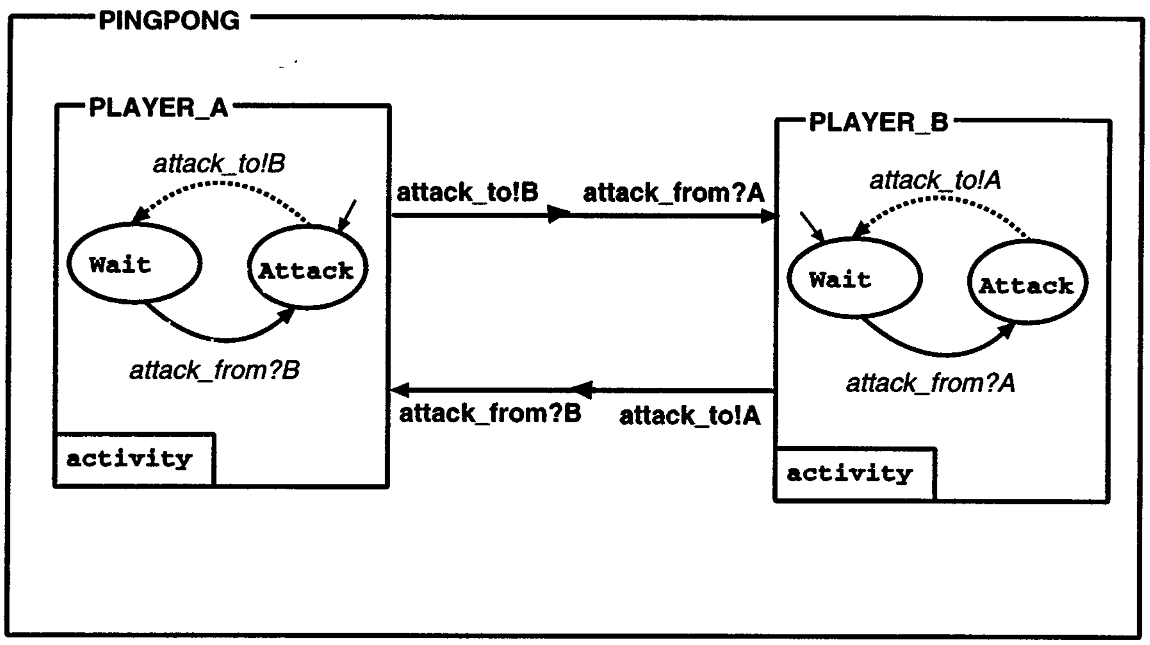
$\delta_{int}(Attack) = Wait$

$ta(wait) = \infty$

$ta(attack) = \text{ATTACK\_TIME}$

$\lambda(Attack) = attack\_to!opponent$

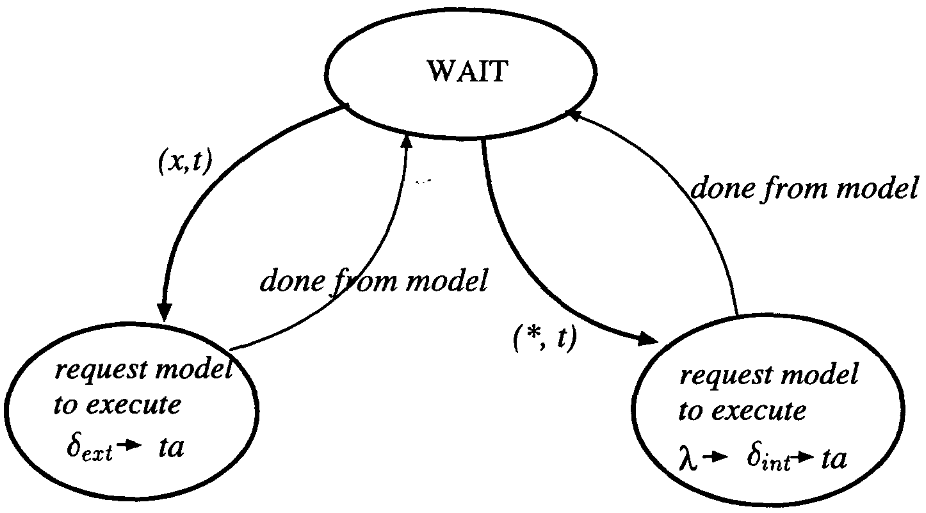
Coupled Model of Ping Pong



$$PINGPONG =< X, Y, M, EIC, EOC, IC, SELECT >$$

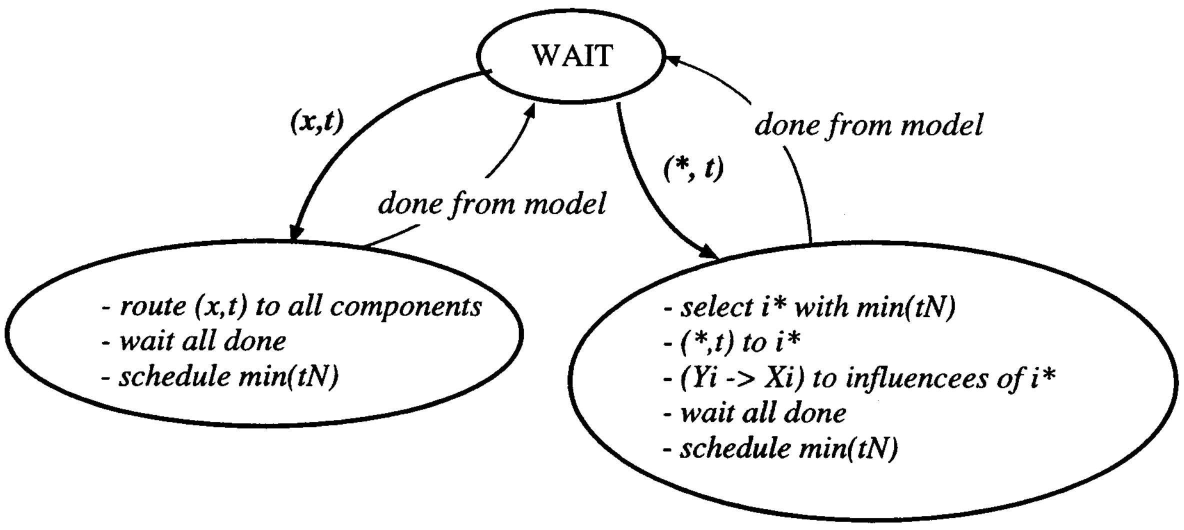
$X = \emptyset$
$Y = \emptyset$
$M = \{PLAYER\_A, PLAYER\_B\}$
$EIC = \emptyset$
$EOC = \emptyset$
$IC = \{(PLAYER\_A.attack\_to!B, PLAYER\_B.attack\_from?A),$
$\quad (PLAYER\_B.attack\_to!A, PLAYER\_A.attack\_from?B)\}$

Abstract Simulator for Atomic DEVS



```
When receive an input $(x, t)$
    done := false
        if $t_L \le t \le t_N$ then
            $e := t - t_L$
            $s := \delta_{ext}(s, e, x)$
            $t_L := t$
            $t_N := t_L + ta(s)$
        else error
    done := true
end when

when receive an input $(*, t)$
    done := false
        if $t = t_N$ then
            $Y := \lambda(s)$
            $s := \delta_{int}(s)$
            $t_L := t$
            $t_N := t_L + ta(s)$
        else error
    done := true
```

Abstract Simulator for Coupled DEVS



```
When receive an input (x,t)
   done := false
      if $t_L \leq t \leq t_N$ 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 send 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
```

DEVSIM++

- Developed at CORE Lab., EE Dept., KAIST

- Discrete Event System Modeling/Simulation Environment

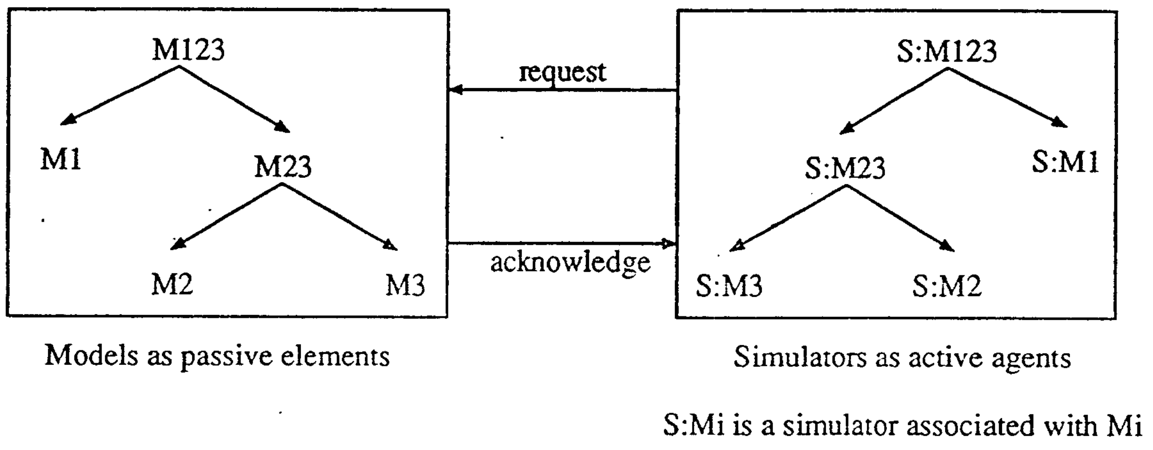
- Realize the DEVS Formalism

- Realize the Abstract Simulator
  - Interprete the Dynamics of DEVS Models
  - Simulators for Atomic Models
  - Coordinators for Coupled Models

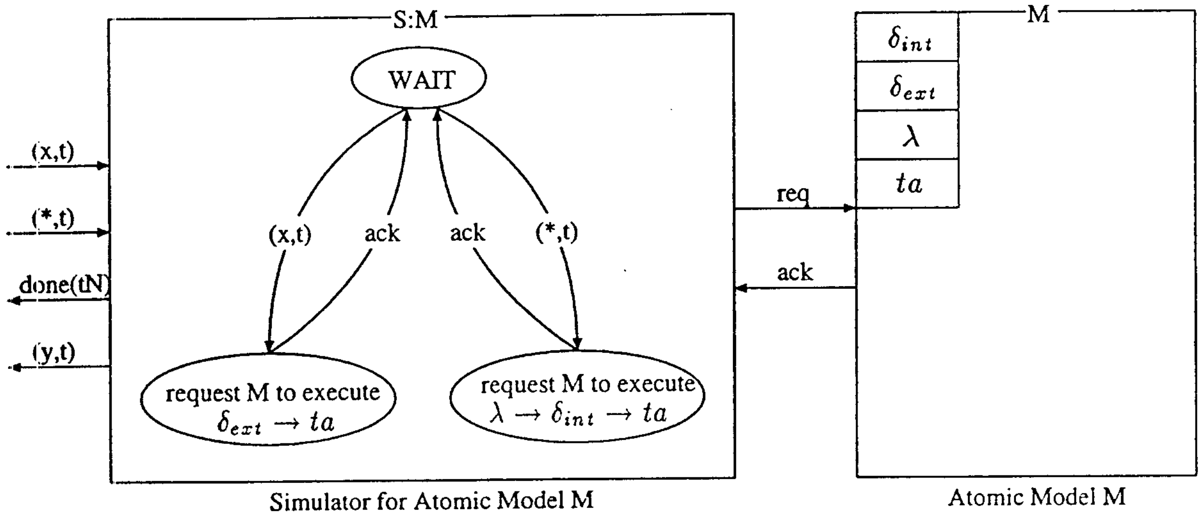
- Object-Oriented Environment Using C++
  - Expressive Power
  - Execution Speed

- Modular and Hierarchical Systems Specification

- Used in Computer Architecture and Computer Network Courses

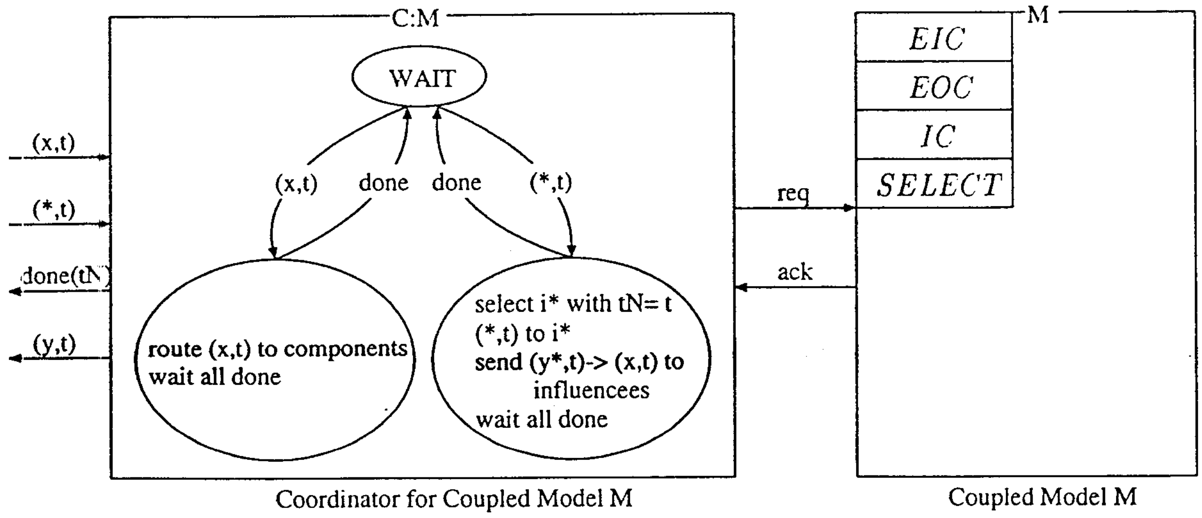

M123
M1
M23
M2
M3
request
acknowledge
S:M123
S:M23
S:M1
S:M3
S:M2
Models as passive elements
Simulators as active agents
S:Mi is a simulator associated with Mi

Passive Models and Active Simulators.

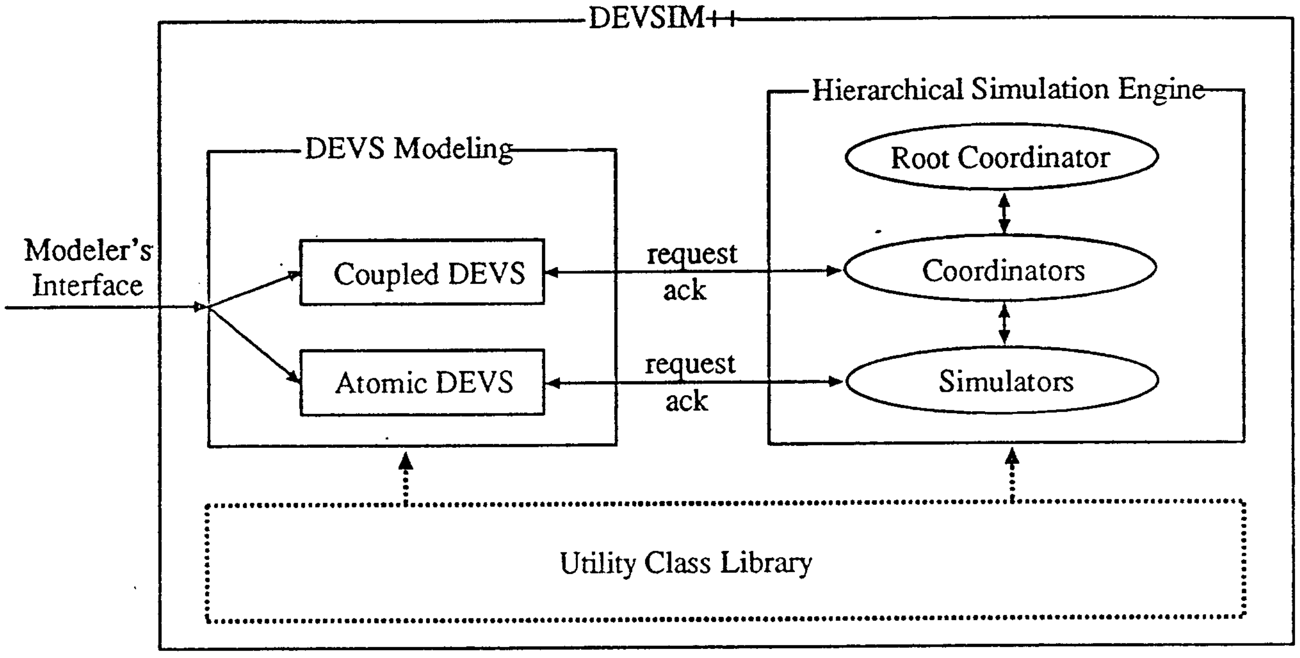


S:M
WAIT
(x,t)
ack
ack
(*,t)
request M to execute $\delta_{ext} \rightarrow ta$
request M to execute $\lambda \rightarrow \delta_{int} \rightarrow ta$
(x,t)
(*,t)
done(tN)
(y,t)
req
ack
M
δ_{int}
δ_{ext}
λ
ta
Simulator for Atomic Model M
Atomic Model M

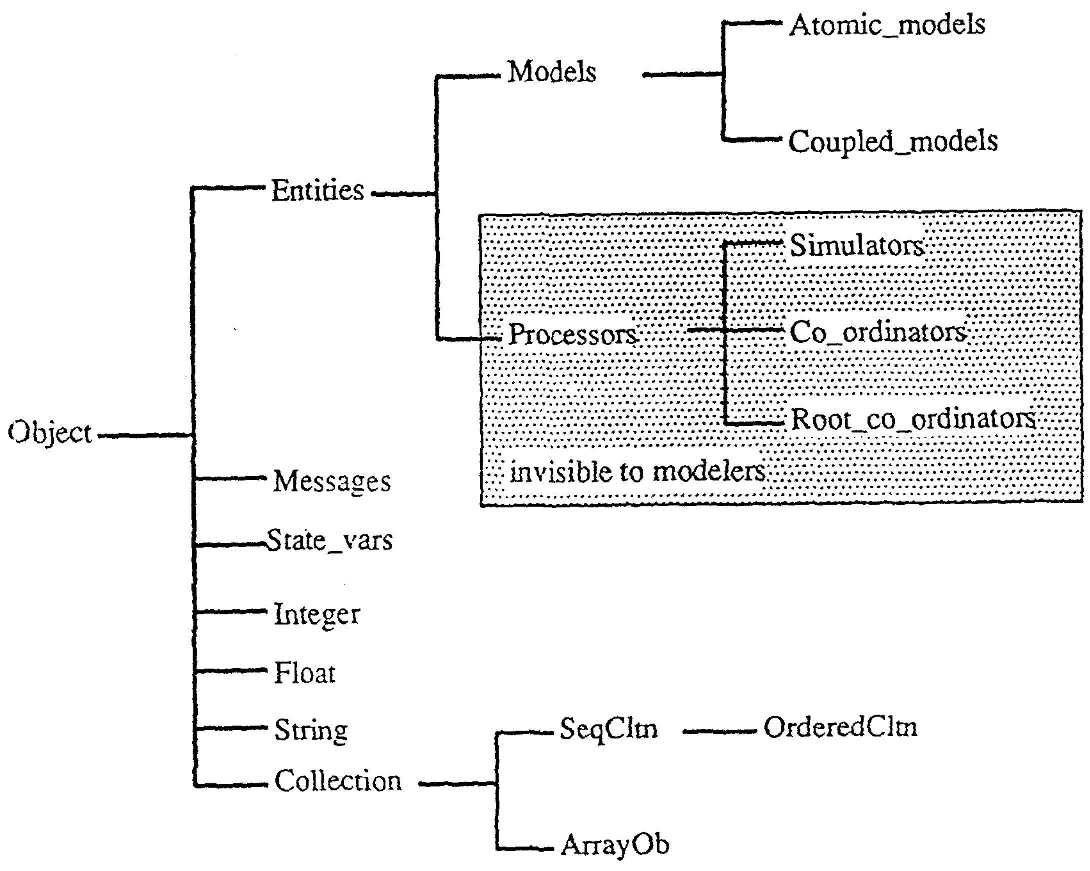
(a) Behavior Outline of Simulator.



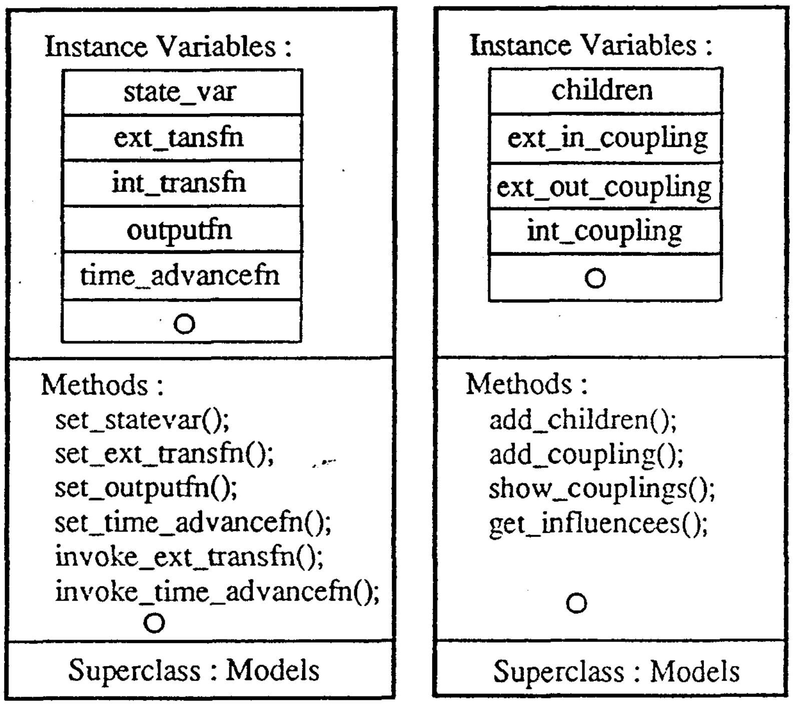
(b) Behavior Outline of Coordinator.



DEVSIM++ Architecture Overview.



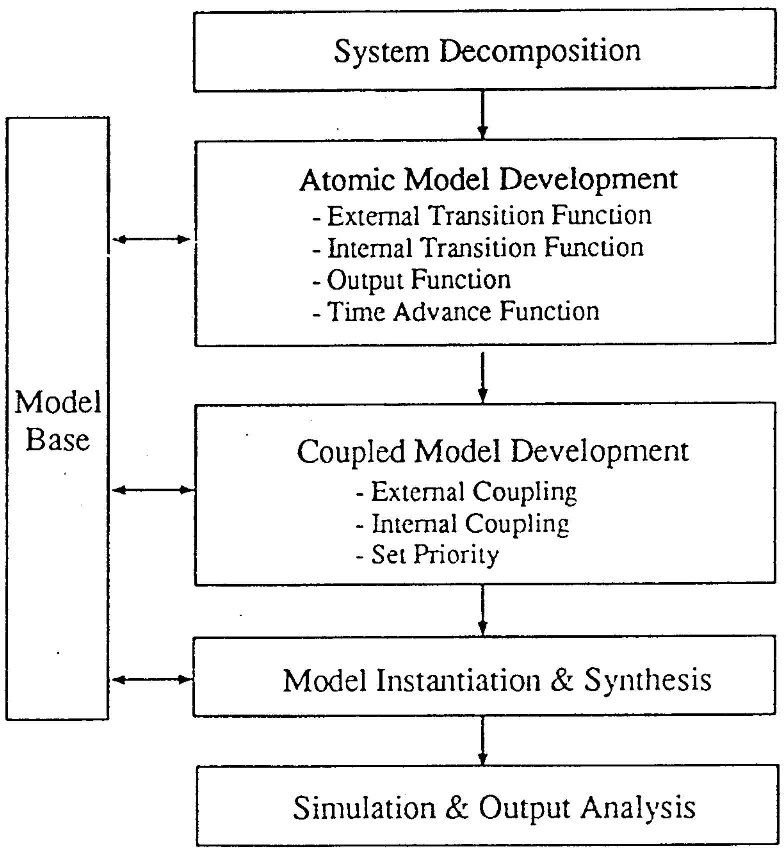
Class Hierarchy of DEVSIM++.

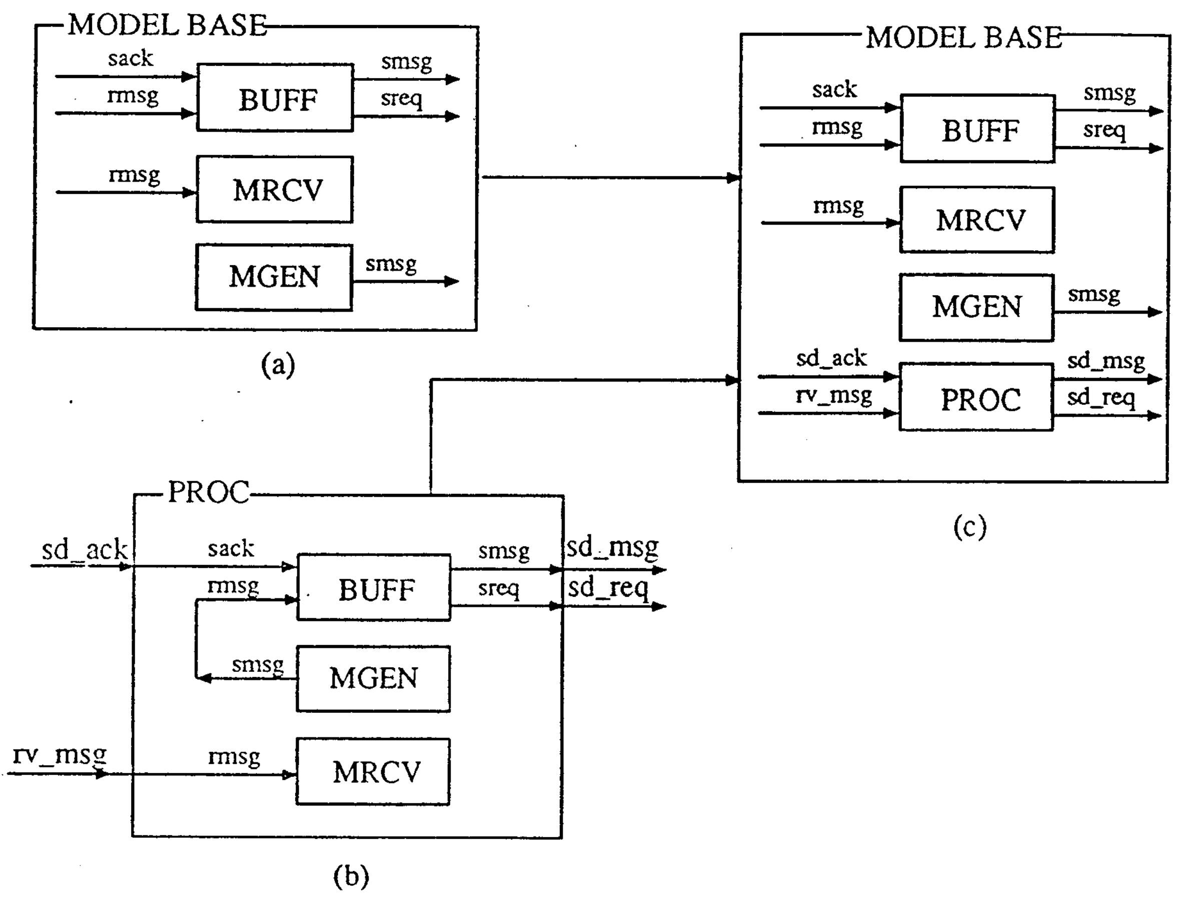


(a) Class Atomic_models. (b) Class Coupled_models.

Definitions for Atomic_models and Coupled_models.

DEVSIM++에서 모델링 및 시뮬레이션





CS of PROC = (EIC, EOC, IC)

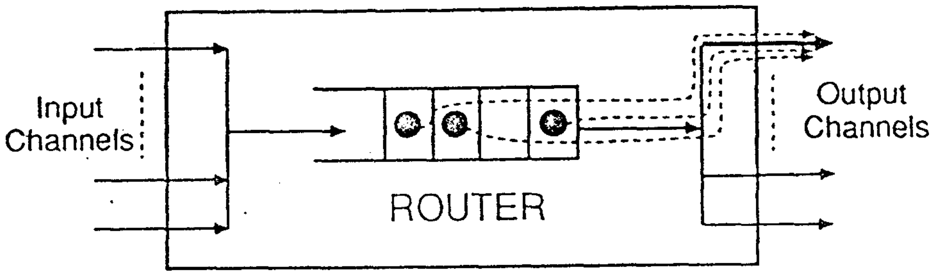
EIC = { (PROC.sd_ack, BUFF.sack) (PROC.rv_msg, MRCV.rmsg) }
EOC = {(BUFF.smsg, PROC.sd_msg) (BUFF.sreq, PROC.sd_req) }
IC = {(MGEN.smsg, BUFF.rmsg)}

(d)

Model Base Concept.

Channel Conflicts in Multiprocessor Realtime Systems

• Channel Conflicts

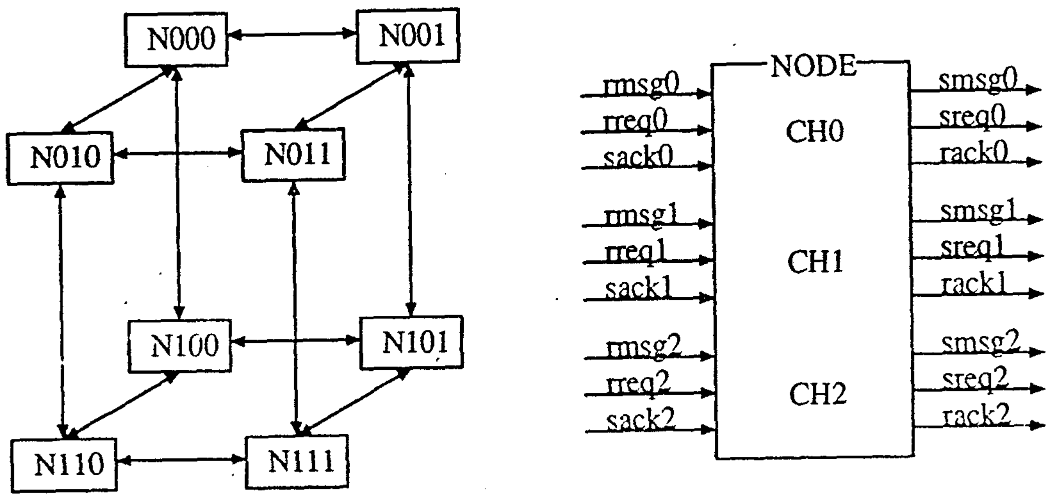


◦ Channel Conflict Resolving Policies

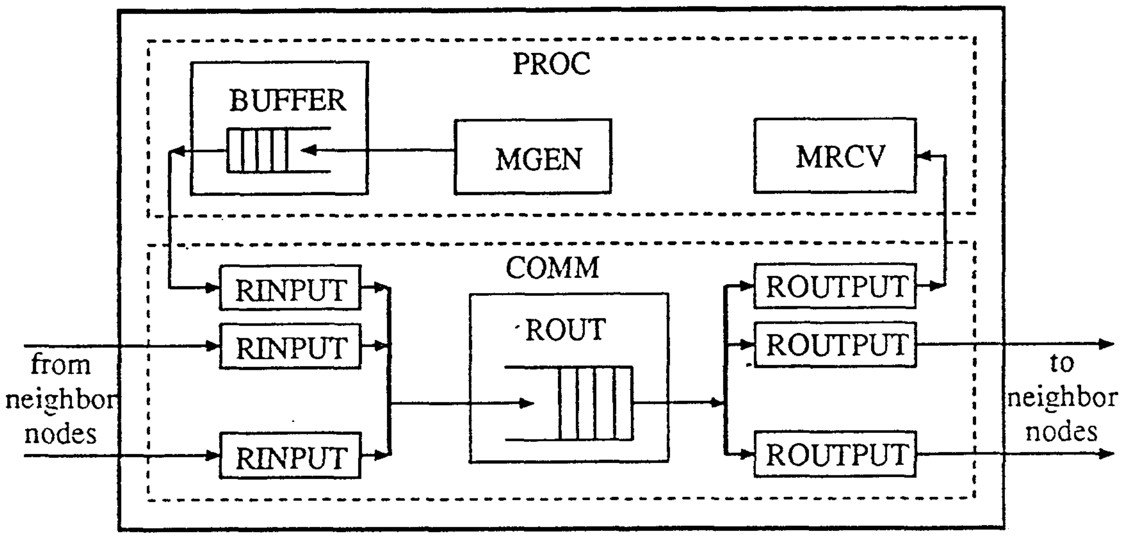
- MDF : Minimum-Dealine-First
- MLF : Minimum-Laxity-First
- MFF : Most-Farthest-First

• Performance Evaluation Through Discrete Event Simulation

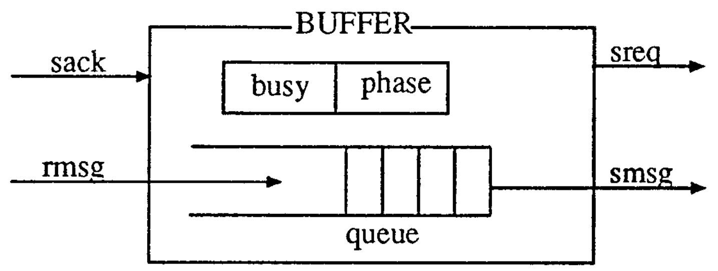
$$\text{message loss ratio} = \frac{\text{total number of messages lost}}{\text{total number of messages generated}}$$



3-dimensional Hypercube and Channel Interface.

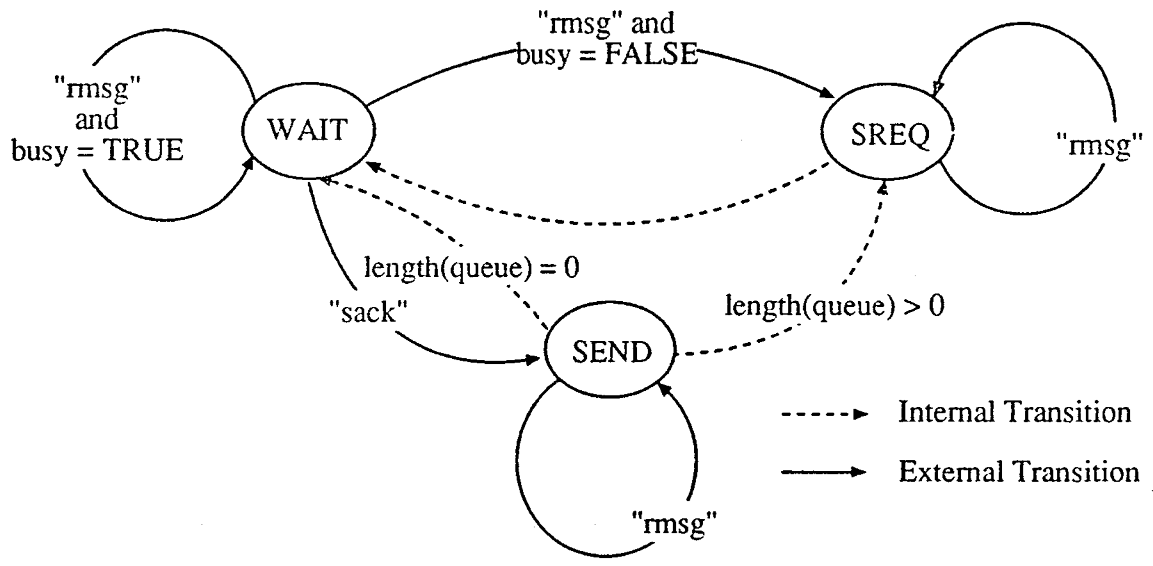


Model of a Node Computer.



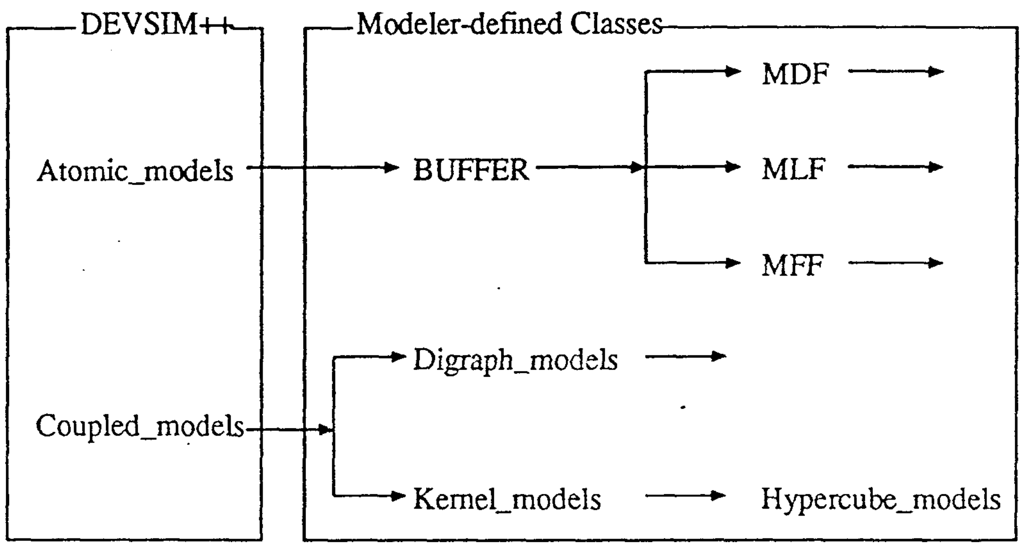
phase : one of {SEND, SREQ, WAIT};
busy : TRUE or FALSE;
queue : priority queue.

(a) I/O Ports and State Variables.



(b) Phase Transitions Diagram.

Atomic Model BUFFER.



Modeler-defined Classes for Modeling of a Hypercube Computer.

```
// External Transition Function //
when receive an input(x,t)
    case input port of :
        "sack"
            phase := SEND;
        "rmsg" :
            insert(x,queue);
            if (phase = WAIT and busy = FALSE) phase := SREQ;
            else continue;

// Internal Transition Function //
case phase of :
    "SEND" :
        delete(first,queue);
        busy := FALSE;
        if (length(queue) = 0) phase := WAIT;
        else phase := SREQ;
    "SREQ" :
        phase := WAIT;
        busy := TRUE;

// Output Function Function //
case phase of:
    "SREQ" :  sreq := 1;
    "SEND" :  smsg := first(queue);

// Time Advance Function //
case phase of:
    "SREQ" :  ta(s) := ReqTime;
    "SEND" :  ta(s) := mem_access_time(first(queue));
    "WAIT" :  ta(s) := INFINITY
```

BUFFER Pseudo Code in DEVS Formalism.

```
// internal transition function //
void BUFFER_int_transfn(State_vars& s)
{
    if (s.get_value("phase") == SEND) {
        s.set_value("busy",FALSE);
        s.get_value("queue") -> delete_msg();
        if (s.get_value("queue")->isEmpty())
            s.set_value("phase",WAIT);
        else
            s.set_value("phase",SREQ);
    } else if (s.get_value("phase") == SREQ) {
        s.set_value("phase",WAIT);
        s.set_value("busy",TRUE);
    }
}

// output function //
void BUFFER_outputfn(const State_vars& s)
{
    if (s.get_value("phase") == SREQ)
        message.set("sreq",TRUE);
    else if (s.get_value("phase") == SEND)
        message.set("smsg",s.get_value("queue")->get_msg());
}

// time advance function //
timeType BUFFER_time_advancefn(const State_vars& s)
{
    if (s.get_value("phase") == SEND)
        return s.get_value("queue") -> mem_access_time();
    else if (s.get_value("phase") == SREQ)
        return ReqTime;
    else
        return INFINITY;
}
```

DEVSIM++ Code for Atomic Model BUFFER.

```
// define new class for atomic model BUFFER
class BUFFER : public Atomic_models {
public :
    BUFFER(const char* name) :  Atomic_models(name) {
        // define I/O ports and state variables//
        add_inports(2,"rmsg","sack");
        add_outports(2,"smsg","sreq");
        set_state_var(3,"busy","phase","queue");

        // initialize the state variables//
        set_state_value("phase",WAIT);
        set_state_value("busy",FALSE);
        set_state_value("queue",new msgq());

        // set characteristic functions //
        set_int_transfn(BUFFER_int_transfn);
        set_outputfn(BUFFER_outputfn);
        set_time_advancefn(BUFFER_time_advancefn);
    }
};
```

Class Definition of Atomic Model BUFFER.

```
// define new class for atomic model BUFFER
class MDF : public BUFFERS {
public :
    MDF(const char* name) :  BUFFER(name) {
        set_ext_transfn(MDF_ext_transfn);
    }
};

// external transition function //
void MDF_ext_transfn(State_vars& s, const timeType& e,
                             const Messages& message)
{
    if (messge.get_port() == "sack")
        s.set_value("phase",SEND);
    else if(messge.get_port() == "rmsg") {
        s.get_value("queue") -> InsWithDeadline(message.get_value());
        if (s.get_value("phase") == WAIT && s.get_value("busy") == FALSE)
            s.set_value("phase",SREQ);
        else
            CONTINUE();
    }
}
```

DEVSIM++ Code for Atomic Model MDF.

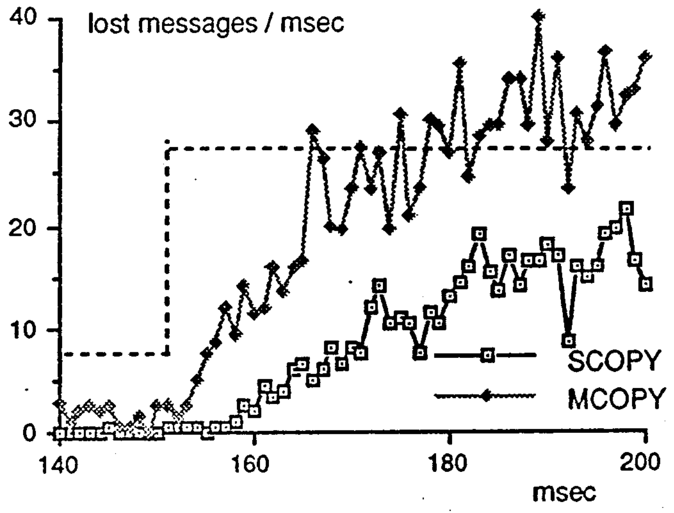
```
Digraph_models PROC = *(new Digraph_models("PROC"));

// define components models and I/O ports //
PROC.add_children(3,mgen,mrcv,mdf);
PROC.add_inports(2,"rv_msg","sd_ack");
PROC.add_outports(2,"sd_msg","sd_req");

// input and output ports couplings //
PROC.add_coupling(PROC,"rv_msg",mrcv,"rmsg");
PROC.add_coupling(PROC,"sd_ack",mdf,"sack");
PROC.add_coupling(mdf,"smsg",PROC,"sd_msg");
PROC.add_coupling(mdf,"sreq",PROC,"sd_req");

// internal couplings between component models //
PROC.add_coupling(mgen,"smsg",mdf,"rmsg");
```

DEVSIM++ Code for Coupled Model PROC.

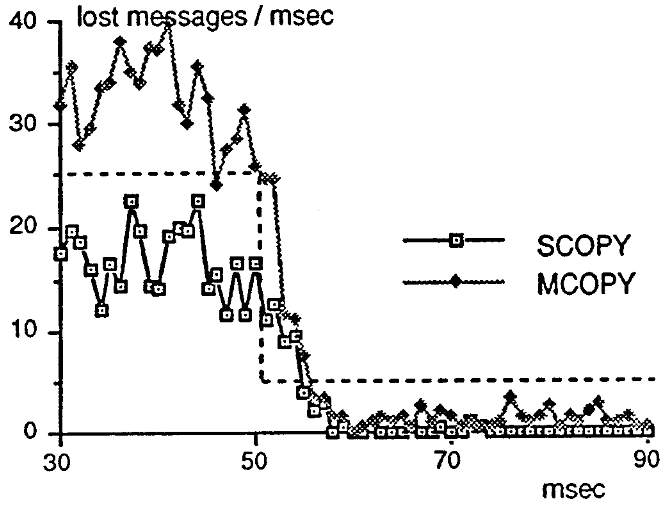
```
Hypercube_models CUBE = *(new Hypercube_models("CUBE"));
// define components models;//
CUBE.make_member(8,node);

// channel interface between node computers //
CUBE.set_channel("smsg","rmsg");
CUBE.set_channel("rack","sack");
CUBE.set_channel("sreq","rreq");
```

DEVSIM++ Code for Hypercube Model CUBE.

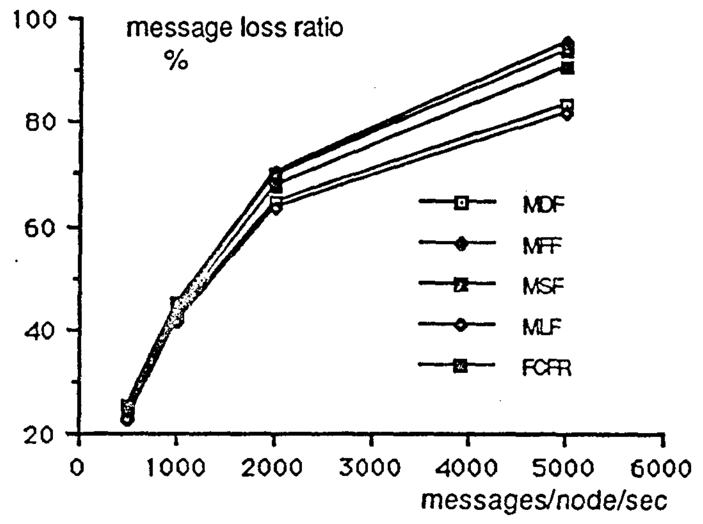
System Dynamics in a Changing Environment : Low-to-High Traffic Load Change



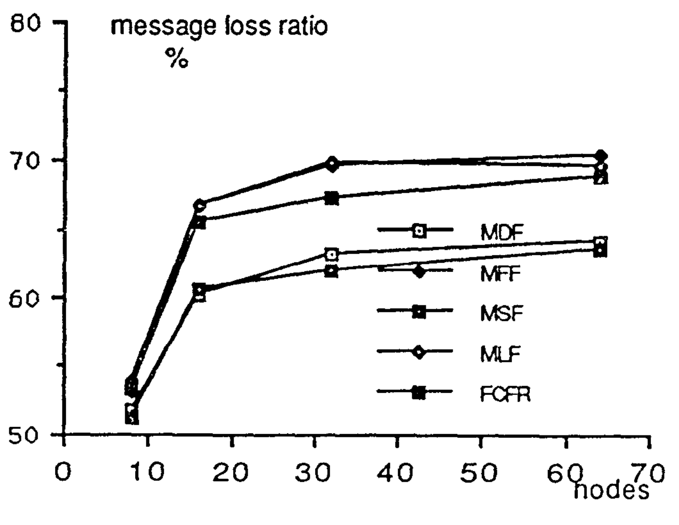
System Dynamics in a Changing Environment : High-to-Low Traffic Load Change

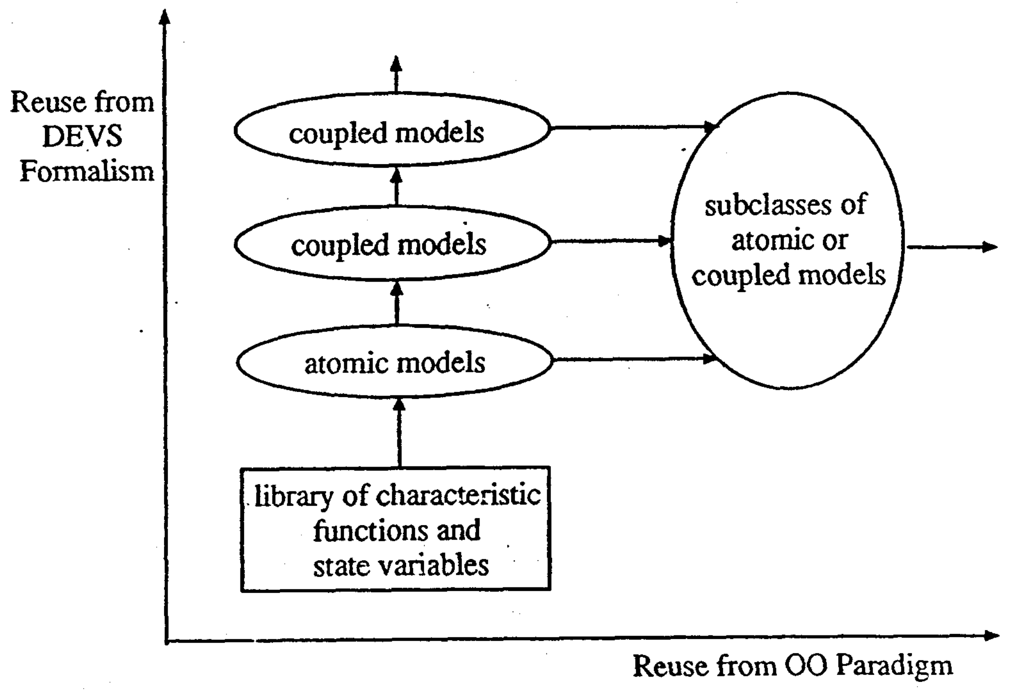


Message Loss Ratio in a 6-cube for Varying Traffic Load



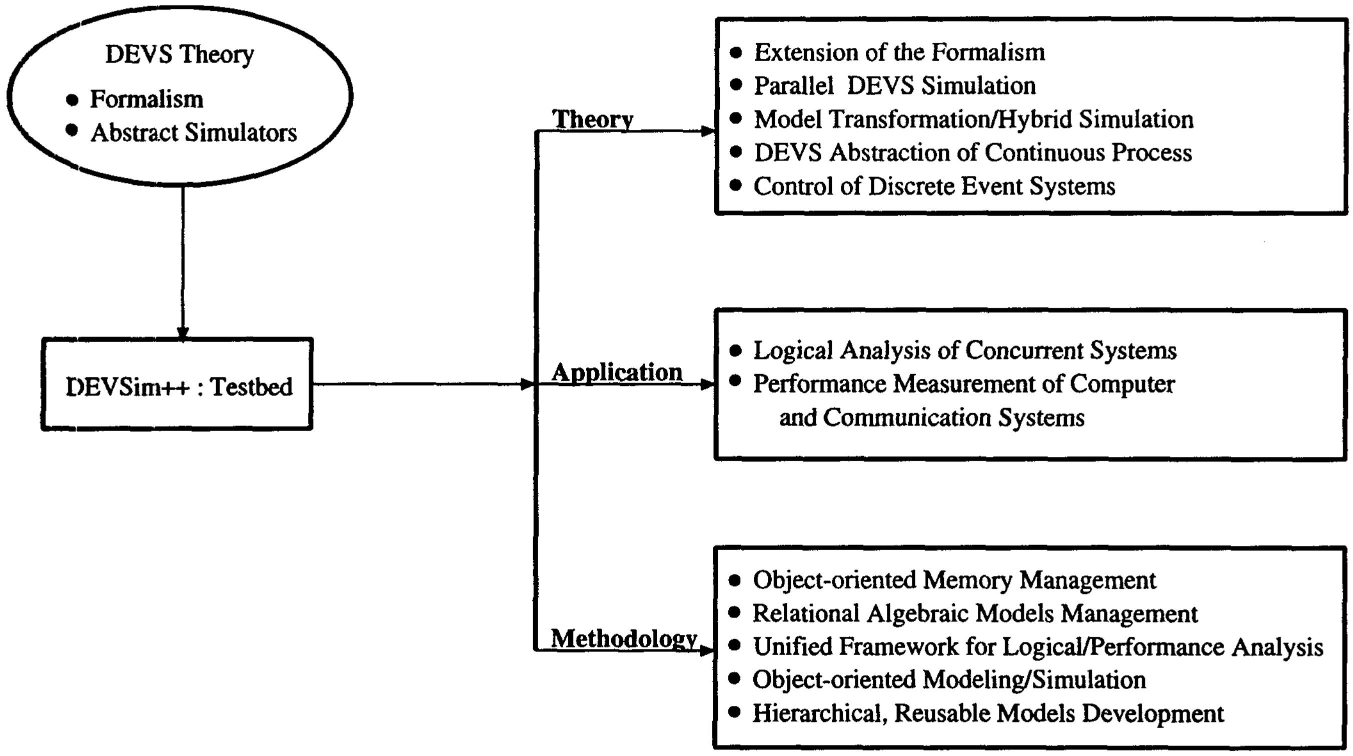
Performance Comparisons for Different Node Sizes




Reuse from DEVS Formalism
coupled models
coupled models
atomic models
library of characteristic functions and state variables
subclasses of atomic or coupled models
Reuse from OO Paradigm

Reusability of Simulation Models in a Two Dimension.

DEVS Research Directions *in CORE*



DEVS Theory
• Formalism
• Abstract Simulators
DEVSim++ : Testbed
Theory
Application
Methodology
• Extension of the Formalism
• Parallel DEVS Simulation
• Model Transformation/Hybrid Simulation
• DEVS Abstraction of Continuous Process
• Control of Discrete Event Systems
• Logical Analysis of Concurrent Systems
• Performance Measurement of Computer and Communication Systems
• Object-oriented Memory Management
• Relational Algebraic Models Management
• Unified Framework for Logical/Performance Analysis
• Object-oriented Modeling/Simulation
• Hierarchical, Reusable Models Development

DEVSim++ : C++-based Modeling and SImulation Environment for DEVS Models

- released in public domain : anonymous ftpable from sim.kaist.ac.kr
- filename : pub/devsimg++.tar.Z

References

1. Books on DES Modeling and Analysis

Zeigler, B.P., *Multifacetted Modeling and Discrete Event Simulation,* Academic Press, 1984.
Zeigler, B.P., *Object-Oriented Simulation with Hierarchical, Modular Models,* Academic Press, 1990.
Cassandras, C.G., *Discrete Event Systems : Modeling and Performance Analysis,* Richard D. Irwin, Inc., and Aksen Associates, Inc., 1993.

2. Journals Focusing on DES

ACM Trans on Modeling and Computer Simulation
(By ACM)
Discrete Event Dynamic Systems : Theory and Applications
(By Kluwer Academic Publishers)
Simulation
(By The Society for Computer Simulation International)
International Journal in Computer Simulation
(By Ablex Publishing Corporation)

3. Journals Dealing with DES

IEEE Trans Systems, Man and Cybernetics
IEEE Trans Automatic Control
IEEE Trans Software Engineering
IEEE Trans Computers
IEEE Trans Communications

4. Good Special Issue on DES Modeling, Simulation and Control

Proceedings of The IEEE, Jan 1989

5. Conferences Dealing with DES Modeling, Simulation and Control

Summer Computer Simulation Conf (every July organized by SCS)
Winter Computer Simulation Conf (every Dec organized by SCS)
IEEE International Sym on Intelligent Control
AI, Simulation and Planning in High Autonomy Systems (every year)