TIMER EMBEDDED FINITE STATE MACHINE MODELING AND ITS APPLICATION

Duckwoong Lee, Byoung K. Choi and Joohoe Kong
Department of Industrial and Systems Engineering
KAIST
335 Gwahak-ro, Yuseong-gu, Daejeon, 305-701, Republic of Korea
E-mail: ldw721@kaist.ac.kr

KEYWORDS

Timer Embedded, Finite State Machine, Synchronization Manager, TEFSM Toolkit

ABSTRACT

In this paper, we propose an extension of classical finite state machine as it called a timer embedded finite state machine (TEFSM) with its formal modeling methods. In the proposed state-based approach, a discrete event system is modeled as a coupled TEFSM. Also presented is a systematic procedure and architecture of developing a simulation executor with a synchronization manager for the coupled TEFSM model. A TEFSM toolkit for modeling and simulation of the proposed TEFSM model has been implemented and a ping pong system was developed as an illustrative example.

INTRODUCTION

A **finite state machine (FSM)** is the oldest known formal model for modeling the sequential behavior of a discrete event system (Wagner 2004). FSM is also called *finite state automata*, finite state transducer, state machine, etc. A ***FSM*** is defined as a model of computations consisting of a set of *states*, a *start state*, an *input alphabet*, and a *transition function* that maps inputs and current states to a *next state*. Computation begins in the start state with an input string and changes to new states depending on the transition function (Paul 2009).

There exist various *formal definitions* of FSM. In a *classical definition* (Peterson 1981), a FSM is defined as a structure (S, X, Y, δ, λ). In *computer science* where the term "finite state automata (FSA)" is mostly used for FSM (Hopcroft 2006), a FSM is defined as a structure (S, X, δ, s_0, F). Also, in *engineering*, a FSM generating outputs is referred to as a *finite state transducer* which is a structure (S, X, Y, δ, s_0, λ), where:

- S is a finite set of states
- X is a finite set of symbols (input alphabet)
- Y is a finite set of symbols (output alphabet)
- δ is the state transition function
- λ is the output function
- s_0 is the start state
- F is the set of final states

Outputs of FSM are generated by *actions*. There are three types of actions associated with a state: (1) ***entry actions*** performed when entering the state, (2) ***exit actions*** performed when exiting the state, and (3) ***input actions*** performed depending on the present state and input conditions. And, an action associated with a transition is referred to as ***transition action*** (Wagner 2003).

Figure 1 shows the execution flow of FSM (Wagner 1992). It waits for an input at the current state, and when the input is received the *input action condition* is tested. If the condition is met, the input action is executed and transition condition is checked. If the transition condition is met, the FSM exits from the current state after executing the *exit action*, moves to the next state while executing the *transition action*, and enters into the next state while executing the *entry action*.

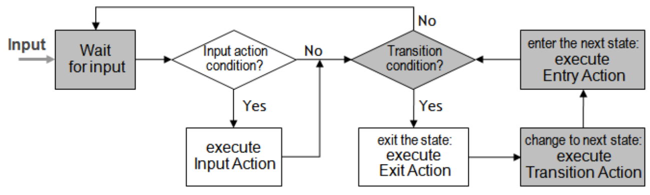

Input
Wait for input
Input action condition?
No
Yes
execute Input Action
Transition condition?
No
Yes
exit the state: execute Exit Action
change to next state: execute Transition Action
enter the next state: execute Entry Action

Figure 1: Execution Flow of FSM with Actions

In this paper, we propose an extension of the FSM which a ***timer*** is embedded as it called the **timer embedded finite state machine (TEFSM)**. The proposed TEFSM is executed with only shading flow in Figure 1 due to the timer. This paper presents a formal modeling definition of the proposed TEFSM and a systematic procedure and architecture of developing a simulation executor with an

Proceedings 24th European Conference on Modelling and Simulation ©ECMS Andrzej Bargiela, Sayed Azam Ali David Crowley, Eugène J.H. Kerckhoffs (Editors)
ISBN: 978-0-9564944-0-5 / ISBN: 978-0-9564944-1-2 (CD)

illustrative example.

TIMER EMBEDDED FSM: TEFSM

We propose a timer embedded FSM (TEFSM) to be used for state-based modeling of discrete event systems. A **timer** means delay time (i.e., timeout value) of a state and then, the state transition occurs automatically after timeout unless it receives further inputs before timeout. An *automatic transition* of states is referred to as an **internal transition**. The proposed TEFSM is *deterministic* (i.e., it has to be in one and only one state at a time), and it has *transition actions* as well as *entry actions*. Also incorporated are *condition* and *probability* associated with transitions, making it a *probabilistic* FSM. In addition, it is allowed to have ***state variables*** which are updated by actions and are used in defining conditions. In summary, the TEFSM has the following extended features:

(1) *Timers* (time-delays) and *entry actions* (E-Action) associated with states.

(2) *Conditions* (Boolean or *probability*) and *actions* (T-Action) associated with transitions.

(3) *Internal transitions* enabled by timers/conditions without external inputs.

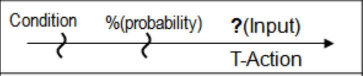
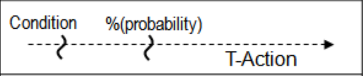
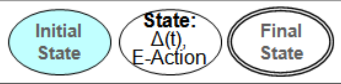
(4) *Time* is implicitly associated with inputs and outputs.

(5) *State variables* are used to reduce the state space.

Specifying a FSM as an *algebraic structure* with a detailed description of the transition function δ is both tedious and hard to read, thus there are two preferred notations for describing FSM (Hopcroft 2006): *FSM diagram* and *state transition table*. Therefore, a **TEFSM diagram** and/or a **state transition table (STT)** which is a tabular listing of the transition function are used in the proposed TEFSM model. The conventions for constructing the proposed **TEFSM diagram** are summarized in Table 1. A number of symbols are used in order to increase readability: '?' symbol for input; '!' for output; '%' for probability; '~' for condition and probability, and 'Δ' for time delay.

In the proposed state-based approach, a discrete event system is modeled as a ***coupled*** TEFSM model consisting of a number of ***atomic*** TEFSM models at modeling phase and the modeling methods with an illustrative example are described in next Section. Also, a *TEFSM executor* may easily be written from the STT of the TEFSM at execution phase and the details of execution of TEFSM are presented in Section 4.

Table 1: Conventions for Constructing a TEFSM Diagram

Primitives	Notations
External Transition Edge	
Internal Transition Edge	
State Node	

T-Action= {!(output), State variable updates}

SYSTEM MODELING EXAMPLE WITH TEFSM

In order to understand our proposed TEFSM easily, a ping pong game, as a well acquainted example, is modeled. Two players, Player-A and Player-B, and an Umpire are involved the game. The rules of the ping pong game with an expedite system (ITTF 2001) are presented in Appendix-A and the reference model is depicted in Figure 2. The expedite system is applied in which an umpire sends an 'Expedite' output if the game is unfinished after 10 minutes, while a game is finished in which any player who wins the game sends an 'Over' output. During a rally, the player sends 'Ball' or 'Out' output.

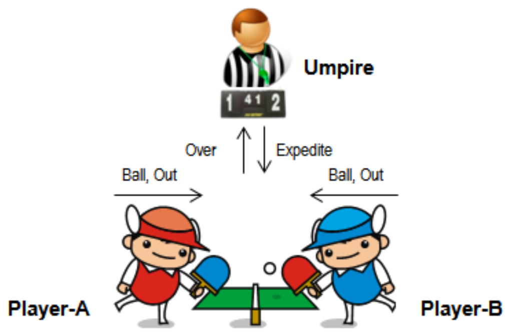


Figure 2: Reference Model of Ping Pong Game

In accordance with conventions in Table 1, we build each atomic TEFSM diagram for two players and an umpire (i.e., atomic TEFSM diagrams for both Player-A and Player-B are identically same except for message name of "Ball" and "Out"). Then put the three diagrams together, a coupled TEFSM model of the ping pong game may be obtained as shown in Figure 3. Since the TEFSM model is self-explanatory, additional explanations are omitted. The internal transition conditions associated with the states and functions are defined in Appendix-B, where t_o = offense delay time, t_w

= wait delay time, t_e = expedited delay time (in Player), t_e = expedited time (in Umpire), P_{IN} = probability of ball-in (success), and P_{OUT} = probability of out (failure).

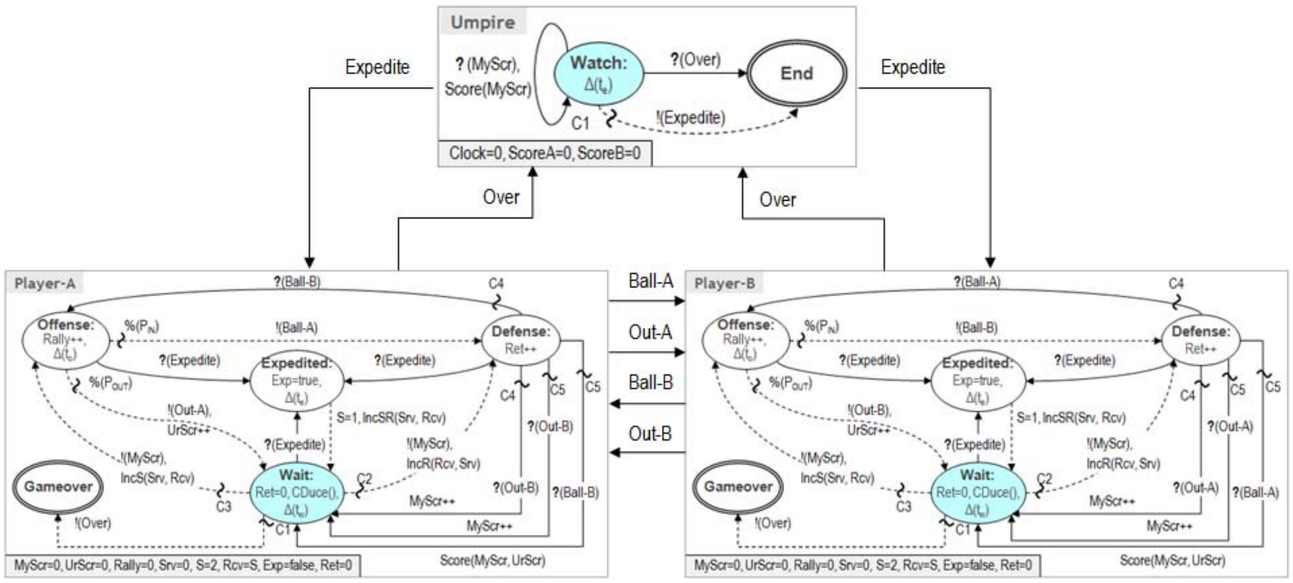


Figure 3: TEFSM Coupled Model of Ping Pong Game

Table 2 is a STT of an atomic TEFSM model for the Player-A. Both TEFSM diagram and their corresponding STT are contained identical model of TEFSM, and therefore the STT of Umpire is omitted.

Table 2: State Transition Table of Player-A

State	Entry Action	Input/Delay	Condition	Transition Action	Next State
Wait	Ret=0, CDuce()	$\Delta(t_w)$	C1	!(Over)	Gameover
			C2	IncR(Rcv, Srv)	Defense
			C3	IncS(Srv, Rcv)	Offense
		?(Expedite)			Expedited
Offense	Rally++	$\Delta(t_o)$	%(P_{IN})	!(Ball-A)	Defense
			%(P_{OUT})	!(Out-A), UrScr++	Wait
		?(Expedite)			Expedited
Defense	Ret++	?(Ball-B)	C4		Offense
			C5	Score(MyScr, UrScr)	Wait
		?(Out-B)	C4	MyScr++	Wait
			C5	MyScr++	Wait
Expedited	Exp=true	$\Delta(t_e)$		S=1, IncSR(Srv, Rcv)	Wait

EXECUTION OF COUPLED TEFSM MODEL

This section presents methods of constructing a TEFSM simulator, which is also a coupled TEFSM, and of building a TEFSM executor of the coupled TEFSM model.

The key issue in the simulation of a coupled TEFSM model is how to synchronize simulation times of individual atomic TEFSM models. The *time-synchronization* (Fujimoto 2000) method we use in this paper is based on the concept of the ***synchronization manager*** (Lee 2010). The overall structure of the TEFSM simulator of Figure 3 is shown in Figure 4.

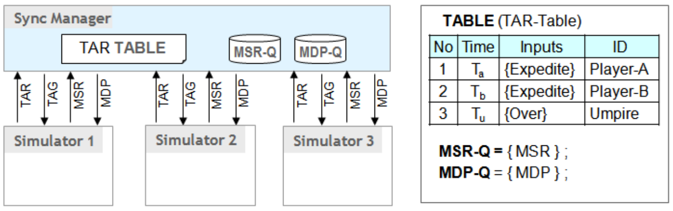


TABLE (TAR-Table)

No	Time	Inputs	ID
1	T_a	{Expedite}	Player-A
2	T_b	{Expedite}	Player-B
3	T_u	{Over}	Umpire

MSR-Q = { MSR } ;
MDP-Q = { MDP } ;

Figure 4: Overall Structure of TEFSM Simulator for the Coupled TEFSM Model in Figure 3

In the TEFSM simulator, all the interactions among the *atomic TEFSM models* are made through ***Sync Manager***. At the beginning, each atomic model sends a TAR (*time advance request*) message to Sync Manager at its *start state*. Then, Sync Manager builds a *TAR table* named TABLE, and sends a TAG (*time advance grant*) message to an atomic model whose request-time is smaller than those of others. An instance of the TAR table is depicted in Figure 4. Upon receiving the TAG, the atomic model advances its simulation time and moves into a new state. During and/or after this *state transition*, the atomic model may send MSR (*message send request*) messages to Sync Manager who will store the received MSR messages in a queue named MSR-Q. Since a MSR in which an "output" of the atomic model is contained may be an "input" to multiple atomic models, the messages to be sent out are temporally stored in another queue named MDP-Q where MDP stands for *massage delivery packet*. The data objects introduced so far have the following structures:

- **TAR = (Time, Inputs, ID)** // time advance request (ID= atomic model ID)
- **TAG = (Now, ID)** // time advance grant (Now= current simulation time)
- **MSR = (Msg, ID)** // message send request (Msg= *input/output* message)
- **MDP = (Msg, Now, ID)** // message delivery packet
- **MSR-Q = {MSR}** // simple list of MSR
- **MDP-Q = {MDP}** // simple list of MDP

Shown in Figure 5 are details of Sync Manager which is also a TEFSM. The synchronization manager presented in Figure 5 is a general one that can be used for any coupled TEFSM model. The functions Select(TAG) is for making a TAG with an atomic model whose TAR is smaller than those of others and Get-MDP(m) is for getting MDPs from MSR-Q in order to deliver them to corresponding atomic models. Also, the model is self-explanatory, the additional explanations are omitted.

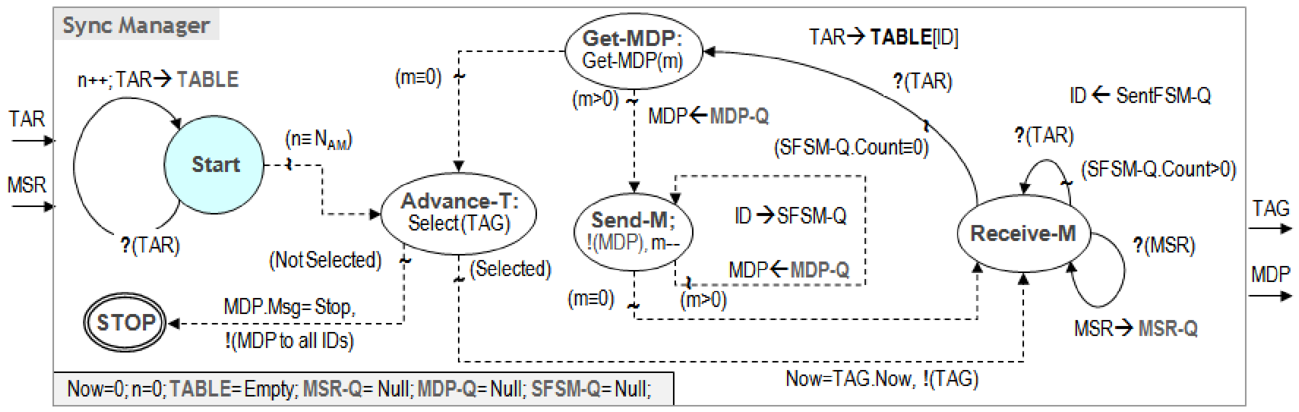


Figure 5: TEFSM Model of the Synchronization Manager in Figure 4

An "atomic model" of a *coupled TEFSM model* (Figure 3) is different from that in a *TEFSM simulator* (Figure 4). The former may be called an *atomic system model* and the latter an *atomic simulation model* or ***atomic simulator*** in short. Therefore, an atomic simulator obtained by converting atomic TEFSM model at execution phase without internal transitions (i.e., this conversion is performed automatically in our TEFSM toolkit which is presented in next Section and makes the TEFSM model into the classical FSM model). In general, the ***rules for converting*** an atomic TEFSM model having final states to an atomic simulator are:

- If the atomic model has *final states*, a STOP state is added as a *final state*.
- Each of the original *final state* is converted to a *regular state*, and a *transition edge* having "?(MDP[Stop])" is defined from each *converted state* to state STOP.
- For all *states* except state STOP, an *entry action* "Clock= Now" is added.
- For a *timed state* with time-out value of t_o, (1) an entry action "!(TAR[t_o, I])" is added where I denotes a set of inputs of the state and (2) each *internal transition* edge is replaced by an *external transition* edge having "?(TAG)".
- For a *state without timer*, an entry action "!(TAR[∞, I])" is added.
- Each output "!(Out)" is replaced by "!(MSR[Out])" and input "?(In)" by "?(MDP[In])."

IMPLEMENTATION AND APPLICATION

Modeler and simulator for the proposed TEFSM model have been implemented as a TEFSM toolkit under a Microsoft .NET Framework 3.5 environment using the C# programming language. The software architecture of TEFSM toolkit is shown in Figure 6.

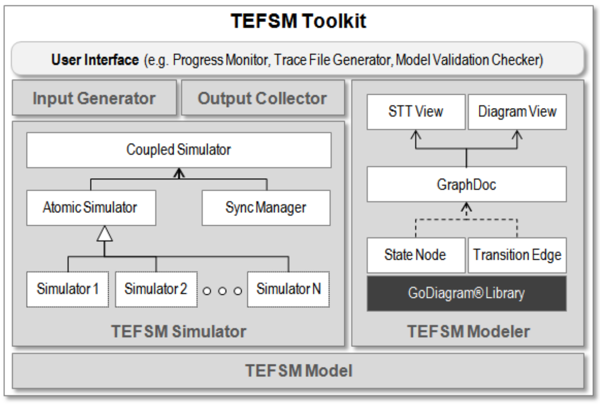
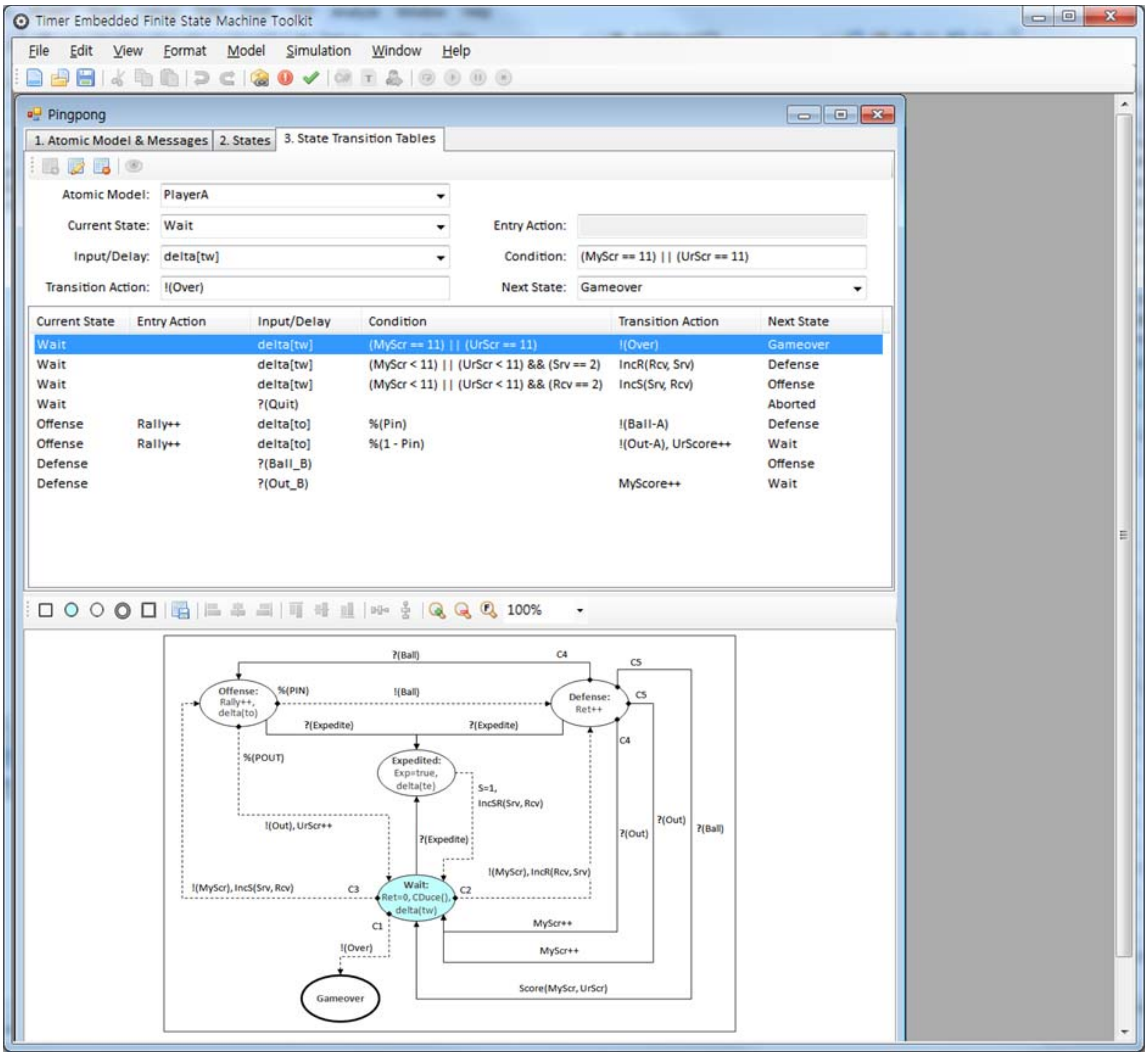


Figure 6: Software Architecture of TEFSM Toolkit

The toolkit consists of a user interface, a TEFSM modeler that provides a diagram view or a STT view of TEFSM model, a TEFSM simulator that converts the TEFSM model into TEFSM simulator and generates C# codes for simulation, and TEFSM model that the fundamental model objects are contained. The installation and tutorial files of the toolkit may be downloaded from http://vms.kaist.ac.kr.

Figure 7 shows a GUI for modeler in the toolkit including the atomic TEFSM model STT (top of Figure 7: same to Table 2) and diagram (bottom of Figure 7: same to one of Figure 3) for Player-A. In this toolkit, the basic C# codes for simulation are generated from the modeler so that the developer could implement the simulator easily.



Atomic Model: PlayerA

Current State: Wait

Input/Delay: delta[tw]

Transition Action: !(Over)

Entry Action:

Condition: (MyScr == 11) || (UrScr == 11)

Next State: Gameover

Current State	Entry Action	Input/Delay	Condition	Transition Action	Next State
Wait		delta[tw]	(MyScr == 11) || (UrScr == 11)	!(Over)	Gameover
Wait		delta[tw]	(MyScr < 11) || (UrScr < 11) && (Srv == 2)	IncR(Rcv, Srv)	Defense
Wait		delta[tw]	(MyScr < 11) || (UrScr < 11) && (Rcv == 2)	IncS(Srv, Rcv)	Offense
Wait		?(Quit)			Aborted
Offense	Rally++	delta[to]	%(Pin)	!(Ball-A)	Defense
Offense	Rally++	delta[to]	%(1 - Pin)	!(Out-A), UrScore++	Wait
Defense		?(Ball_B)			Offense
Defense		?(Out_B)		MyScore++	Wait

Figure 7: GUI of Modeler in TEFSM Toolkit

In order to animate progress of the simulation after or while the simulation is run, the toolkit generates the trace file of the simulation result for Proof® Animation (See http://www.wolverinesoftware.com). Figure 8 shows a screen capture of the ping pong game animation of which a trace file generated from the TEFSM toolkit.

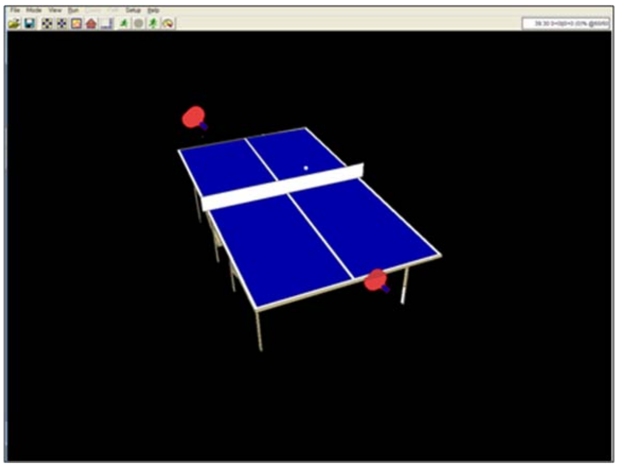


Figure 8: Proof® Animation with TEFSM Trace File

CONCLUSION

Presented in this paper is an extension of FSM, timer embedded FSM (TEFSM). In the proposed TEFSM, a discrete event system is modeled as a coupled TEFSM and well simulated as a simulation executor with synchronization manager. Also presented is a TEFSM toolkit for modeling and simulation with a ping pong game as an illustrative example. Further developments of TEFSM toolkit might be needed in order to test rigorously and undergo a refinement of a real-life environment.

Comparing to the class *discrete event system specification* (DEVS) which is a well known state-based modeling formalism (Zeigler 2000), the proposed TEFSM has generality and modeling power as follows: (1) the output function λ is defined as a *transition action* of an *external transition*, while the output of DEVS is a transition action of an *internal transition*, (2) state variables that are used in defining conditions for transitions are allowed, and (3) supports the entry actions while DEVS does not.

APPENDIX

A. Modeling assumptions (considering rules) of a ping pong game

- Consider only one match with two player and one umpire.
- A game is over without duce if a player scores 11 points (each player servers twice before changing its turn).
- If both players reach 10 points, then service alternates after each point, until one player gains a two point lead.
- Expedite system is as follows:
  - Except where both players have scored at least 9 points, the expedite system shall come into operation if a game is unfinished after 10 minutes' play or at any earlier time at the request of both players.
  - If the ball is in play when the time limit is reached, play shall be interrupted by the umpire and shall resume with service by the player who served in the rally that was interrupted.
  - Thereafter, each player shall serve for 1 point in turn until the end of the game and if the receiving player 13 returns the receiver shall score a point.
  - Once introduced, the expedite system shall remain in operation until the end of the match.

B. Internal transition conditions and details of functions used in Figure 3

```
C1 = ((MyScr ≡ 11) || (UrScr ≡ 11)) & (abs(MyScr, UrScr) ≡
2) //Game-over
C2 = ((MyScr < 11) & (UrScr < 11) || (abs(MyScr, UrScr) < 2))
& (Srv ≡ S) //Receive
C3 = ((MyScr < 11) & (UrScr < 11) || (abs(MyScr, UrScr) < 2))
& (Rcv ≡ S) //Serve
C4 = (Exp ≡ false) || ((Exp ≡ true) & (Ret <13)) //Normal
C5 = (Exp ≡ true) & (Ret ≡ 13) //Expedited & score

IncS(Srv, Rcv): Srv += 1; if(Srv ≡ S) then Rcv = 0
IncR(Rcv, Srv): Rcv += 1 ; if(Rcv ≡ S) then Srv = 0
IncSR(Srv, Rcv): if(Srv ≤ S) then Srv = 0, Rcv = S
                 else then Srv = S, Rcv = 0
Score(MyScr, UrScr): if(Srv ≡ S) then MyScr++;
                     else then UrScr++;
CDuce(): if(MyScr ≡ 9 & UrScr ≡ 9) then S = 1
```

REFERENCES

Fujimoto, R.M., 2000, *Parallel and Distributed Simulation Systems*, John Wiley & Sons

Hopcroft, J.E. *et al.*, 2006, *Introduction to Automata Theory, Languages, and Computation*, 3rd Ed., Addison Wesley

ITTF (International Table Tennis Federation), 2001, http://www.allabouttabletennis.com/basic-table-tennis-rule.html

Lee, D., Shin, H., and Choi, B.K., 2010, Mediator Approach to Direct Workflow Simulation, *Simulation Modeling Practice and Theory*, Volume 18, Issue 5, pp. 650-662

Peterson, J.L., 1981, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall

Paul E. Black, 2009, http://www.itl.nist.gov/div897/sqg/dads/HTML/finiteStateMachine.html

Wagner, F. and Wolstenholme, P., 1992, VFSM Executable Specification, *IEEE CompEuro 1992 Proceedings*

Wagner, F. and Wolstenholme, P., 2003, Modeling and Building Reliable, Re-usable Software, *IEEE ECBS'03 Proceedings*

Wagner, F. and Wolstenholme, P., 2004, Misunderstandings about State Machines, *Computing and Control Eng.* Volume 15, Issue 4, pp.40–45

Zeigler, B., Praehofer, H. and Kim, T., 2000, *Theory of Modeling and Simulation*, Academic Press: Boston

AUTHOR BIOGRAPHIES

Duckwoong Lee is a post doctor in the Department of Industrial and Systems Engineering at KAIST. He received a BS from Ajou University in 2002, a MS from KAIST in 2004, and a Ph.D. from KAIST in 2010, all in Industrial Engineering. His research interests are in the area of business process management system (BPMS), system modeling and simulation, and parallel and distributed simulation.

Byoung K. Choi is a professor of the Department of Industrial and Systems Engineering at KAIST since 1983. He received a BS from Seoul National University in 1973, a MS from KAIST in 1975, and a Ph.D. from Purdue University in 1982, all in Industrial Engineering. His current research interests are system modeling and simulation, BPMS, simulation-based scheduling, and virtual manufacturing. He can be reached at bkchoi@kaist.ac.kr

Joohoe Kong is a graduate student in the Department of Industrial and Systems Engineering at KAIST. She received a BS from Arizona State University in 2005, a MS from KAIST in 2007 in Industrial Engineering. Her research interests are in the area of BPMS, systems modeling and simulation. She can be reached at joohoe@vmslab.kaist.ac.kr