*Proceedings of the 1998 Winter Simulation Conference*
*D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds.*

# ABSTRACT SIMULATORS FOR THE DSDE FORMALISM

Fernando J. Barros

Departamento de Engenharia Informática
Universidade de Coimbra, Pólo II
P-3030 Coimbra, PORTUGAL

## ABSTRACT

We present the DSDE formalism, a methodology for representing discrete event systems that change structure dynamically. We prove that the DSDE formalism is closed under coupling and that it can be used to construct hierarchical and modular models. The abstract simulators necessary to execute dynamic structure models are also presented. Simulators allow a description of models independently of the actual simulation procedure, and thus encourages model reuse.

## 1 INTRODUCTION

Many real systems have the capability to change their structure. A growing field of research has been developing the foundations of a modeling theory of adaptable systems. A comparison of formalisms and simulation environments that support dynamic structures can be found in Barros (1997a).

An extension of General Systems Theory to include dynamic structure systems is described by Barros (1997c), where the concept of Dynamic Structure System Network is applied to a variety of systems families, including differential equations (DSDQ), discrete time (DSDT), and discrete event systems (DSDE).

We describe the DSDE formalism, and its structured version. We present also the abstract simulators necessary to simulate DSDE models. The separation of models from simulators were introduced by Zeigler(1984), and allow a description of models independently of the actual simulation procedure, improving model reuse.

## 2 DISCRETE EVENT SYSTEM SPECIFICATION

The *Discrete Event System Specification* (DEVS) is a formalism introduced by Zeigler (1976) to describe discrete event systems. In the DEVS formalism a basic model is described by

$$M = (X,s_0,S,Y,\delta,\lambda,\tau)$$

where

$X$ is the set of input values
$S$ is the set of partial states
$s_0$ is the initial partial state
$Y$ is the set of output values
$\delta: Q \times X^{\phi} \to S$ is the transition function, where
$X^{\phi} = X \cup \{\phi\}$
$Q = \{(s,e) | \, s \in S, \, 0 \le e \le \tau(s)\}$ is the state set
$e$ is the time elapsed since last transition
$q_0 = (s_0,0)$ is the initial state
$\phi$ is the absence of value
$\lambda: S \to Y$ is the partial output function
$\tau: S \to \mathbf{R}_0^+$ is the time advance function

The output function, $\Lambda: Q \to Y^{\phi}$, is defined by

$$\Lambda(s,e) = \begin{cases} \lambda(s) & \text{if } e = \tau(s) \\ \phi & \text{if } e < \tau(s) \end{cases}$$

If no event arrives to the system it will stay in partial state $s$ for time $\tau(s)$. When $e = \tau(s)$ the system changes to the state $(\delta(s,\tau(s),\phi),0)$. If an external event, $x \in X$, arrives when the system is in the state $(s,e)$ the system changes to the state $(\delta(s,e,x),0)$. If an external event, $x \in X$, arrives when $e = \tau(s)$, the system changes to the state $(\delta(s,\tau(s),x),0)$. A more detailed description can be found in Zeigler (1976), Zeigler (1984), Zeigler (1990), Chow (1994) and Barros (1998).

## 3 STRUCTURED DISCRETE EVENT SYSTEM SPECIFICATION (SDE)

The *Structured Discrete Event System Specification* (SDE) provides a formalism for obtaining a fine-grained control over model components. The transition function, for example, is decomposed into a set of transitions, where each one is used in certain conditions. The same applies to the other functions. In the SDE formalism a model is described by

$$\mathcal{S} = (X,s_0,S,Y,\kappa,I,\{\delta_i \mid i \in I\},\{\lambda_i \mid i \in I\},\{\tau_i \mid i \in I\})$$

where

$\kappa: S \rightarrow I$, is the index function

EQUIVALENCE 1. A SDE model $\mathcal{S} = (X,s_0,S,Y,\kappa,I,\{\delta_i\},\{\lambda_i\},\{\tau_i\})$, is equivalent to the DEVS model $M = (X,s_0,S,Y,\delta,\lambda,\tau)$, where $\delta(s) = \delta_{\kappa(s)}(s)$, $\lambda(s) = \lambda_{\kappa(s)}(s)$, and $\tau(s) = \tau_{\kappa(s)}(s)$, for all $s \in S$.

A structured model can be used in some situations where dynamic structures used to be helpful. Replace models or change model functions can, in many situations, be achieved by structured models.

## 4 PARALLEL DYNAMIC STRUCTURE DISCRETE EVENT SYSTEM SPECIFICATION

The problem of representing discrete event systems that undergo structural changes has been subject of research (Zeigler and Praehofer 1989), (Barros (1997a). Some applications of dynamic structure models can be found in (Zeigler 1991), (Barros 1996; Barros 1997b). A rigorous approach was accomplished by the DSDEVS formalism (Barros 1995; Barros 1996a). This formalism was implemented in the DELTA simulation environment (Barros 1997).

The *Parallel Dynamic Structure Discrete Event System Specification* (DSDE) is a generalization of the DSDEVS formalism, and allows the specification of basic or dynamic structure networks of discrete event systems. A Parallel Dynamic Structure Discrete Event System Network is a 4-tuple

$$DSDEN_N = (X_N,Y_N,\chi,M_\chi)$$

where

$N$ is the network name

$X_N$ is the network input values set

$Y_N$ is the network output values set

$\chi$ is the name of the dynamic network executive

$M_\chi$ is the model of the executive $\chi$

The model of the executive, is a modified basic model and is defined by the 9-tuple

$$M_\chi = (X_\chi,s_{0,\chi},S_\chi,Y_\chi,\gamma,\Sigma^*,\delta_\chi,\lambda_\chi,\tau_\chi)$$

where

$\gamma: S_\chi \rightarrow \Sigma^*$ is the structure function

$\Sigma^*$ is the set of network structures

A structure $\Sigma_\alpha \in \Sigma^*$ associated with the executive partial state $s_{\alpha,\chi} \in S_\chi$, is given by

$$\Sigma_\alpha = \gamma(s_{\alpha,\chi}) = (D_\alpha,\{M_{i,\alpha}\},\{I_{i,\alpha}\},\{Z_{i,\alpha}\})$$

where

$D_\alpha$ is the set of component names associated with the executive partial state $s_{\alpha,\chi}$

for all $i \in D_\alpha$

$M_{i,\alpha}$ is the model of component $i$

for all $i \in D_\alpha \cup \{\chi,N\}$

$I_{i,\alpha}$ is set of components influencers of $i$

for all $i \in D_\alpha \cup \{\chi\}$

$Z_{i,\alpha}$ is the input function of component $i$

$Z_{N,\alpha}$ is the network output function

These variables are subject to the following constraints: for every $s_{\alpha,\chi} \in S_\chi$:

$\chi \notin D_\alpha$

$N \notin I_{N,\alpha}$

for all $i \in D_\alpha$

$M_{i,\alpha} = (X_{i,\alpha},s_{0,i},S_i,Y_{i,\alpha},\delta_{i,\alpha},\tau_i,\lambda_{i,\alpha})$

is a basic model, with $\delta_{i,\alpha}: Q_i \times X^{\phi}_{i,\alpha} \rightarrow S_i$

for all $i \in D_\alpha \cup \{\chi,N\}$

$$Z_{i,\alpha}: \mathop{\times}_{j \in I_{i\alpha}} V_{j,\alpha} \rightarrow X_{i,\alpha}$$

where

$$V_{j,\alpha} = \begin{cases} Y_{j,\alpha} & \text{if } j \neq N \\ X_N & \text{if } j = N \end{cases}$$

$Z_{i,\alpha}(\mathop{\times}_{|I_{i\alpha}|} \phi) = \phi$, for all $i \in D_\alpha \cup \{\chi,N\}$

For construct models in a hierarchical and modular form we need to prove the DSDE formalism is closed under coupling, that is a DSDE network model is equivalent to a basic model.

**EQUIVALENCE 2**. The DSDE network $DSDEN_N = (X_N,Y_N,\chi,M_\chi)$ is equivalent to the structured model $\mathcal{S} = (X,s_0,S,Y,\kappa,I,\{\delta_i\},\{\lambda_i\},\{\tau_i\})$. The equivalence between the structured model $\mathcal{S}$ and the basic model $M = (X,s_0,S,Y,\delta,\lambda,\tau)$ is guaranteed by Equivalence 1.

We show how to construct a structured model from the network $DSDEN_N = (X_N,Y_N,\chi,M_\chi)$. The input set is given by

$$X = X_N$$

We define $C_\alpha$, the set of all the components, including the executive, associated with a state $s_{\alpha,\chi} \in S_\chi$, by

$$C_\alpha = D_\alpha \cup \{\chi\}$$

The initial partial state $s_0$ is given by

$$s_0 = \mathop{\times}_{i \in C_0} q_{0,i}$$

The partial state $S$ is given by

$$S = \bigcup_{s_\alpha \in S_\chi} (\mathop{\times}_{i \in C_\alpha} Q_i)$$

Let the state of the executive be given by $S_\chi = \{s_{0,\chi}, s_{1,\chi}, s_{2,\chi}, \ldots, s_{j,\chi},\ldots\}$, then the set of indexes $I$ is given by

$$I = \{0,1,2,\ldots, j,\ldots\}$$

The index function, $\kappa: S \to I$, is defined by

$$\kappa(s) = \kappa((s_{\alpha,\chi},e_\chi), \ldots) = \alpha$$

The output set is given by

$$Y = Y_N$$

We define $r_i$ the time component $i$ must still remain in the current partial state, by

$$r_i = \tau_i(s_i) - e_i$$

The time advance function at every index $\alpha$ is given by

$$\tau_\alpha: S \to \mathbf{R}_0^+$$

and is defined by

$$\tau_\alpha(s) = \min\{r_i \mid i \in C_\alpha\}$$

The partial output function at index $\alpha$ is defined by

$$\lambda_\alpha(s) = Z_{N,\alpha}(\bigtimes_{i\in I_{N,\alpha}} \Lambda_{i,\alpha}(s_i,e_i + \tau_\alpha(s)))$$

The output function at index $\alpha$ is defined by

$$\Lambda_\alpha(s,e) = Z_{N,\alpha}(\bigtimes_{i\in I_{N,\alpha}} \Lambda_{i,\alpha}(s_i,e_i + e))$$

The set of states $Q$ is given by

$$Q = \{(s,e) \mid s \in S, 0 \le e \le \tau(s)\}$$

The transition function at index $\alpha$, $\delta_\alpha: Q \times X^\phi \to S$, is given by

$$\delta_\alpha(\bigtimes_{i\in C_\alpha} q_i,e,x) = \bigtimes_{j\in C_\beta} q_j$$

To define the transition $\delta_\alpha$ we will now show how to obtain the new partial executive state $s_{\beta,\chi}$ from the previous state $s_{\alpha,\chi}$. The next executive state $q_{\beta,\chi}$ is given by

$$q_{\beta,\chi} = \begin{cases} (s_{\alpha,\chi},e_\chi + e) & \text{if } x_\chi = \phi \wedge r_\chi > e \quad (1) \\ (\delta_\chi(s_{\alpha,\chi},e_\chi + e,x_\chi),0) & \text{if } x_\chi \ne \phi \vee r_\chi = e \quad (2) \end{cases}$$

where

$$x_\chi = Z_{\alpha,\chi}(\bigtimes_{i\in I_{\chi,\alpha}} v_i)$$

with

$$v_i = \begin{cases} \Lambda_\chi(s_i,e_i + e) & \text{if } i \ne N \\ x & \text{if } i = N \end{cases}$$

The new structure associated with the partial state $s_{\beta,\chi}$ is given by

$$\Sigma_\beta = \gamma(s_{\beta,\chi}) = (D_\beta,\{M_{i,\beta}\},\{I_{i,\beta}\},\{Z_{i,\beta}\})$$

To finish the definition of the transition function we need to show how the remaining components are changed. The set of new components is given by $A = C_\beta - C_\alpha$, and the set of kept components is given by $K = C_\alpha \cap C_\beta$. The state $q_i \in Q_i$ of each component $i \in C_\beta$ is given by

$$q_i = \begin{cases} (s_i,e_i + e) & \text{if } i \in K \wedge (x_i = \phi \wedge r_i > e) \quad (1) \\ (\delta_{i,\alpha}(s_i,e_i + e,x_i),0) & \text{if } i \in K \wedge (x_i \ne \phi \vee r_i = e) \quad (2) \\ q_{0,i} \quad \text{if } i \in A & (3) \end{cases}$$

where

$$x_i = Z_{i,\alpha}(\bigtimes_{i\in I_{i,\alpha}} v_i)$$

with

$$v_i = \begin{cases} \Lambda_{i,\alpha}(s_i,e_i + e) & \text{if } i \ne N \\ x & \text{if } i = N \end{cases}$$

Line 1 of the definition computes the next state of the unchanged components. These models only update their elapsed time. Line 2 computes the next state of the models that either receive an external input or are scheduled to change.

All the outputs are taken simultaneously and the current network structure $(D_\alpha,\{M_{i,\alpha}\},\{I_{i,\alpha}\},\{Z_{i,\alpha}\})$, is used with these values. The new structure $(D_\beta,\{M_{i,\beta}\},\{I_{i,\beta}\},\{Z_{i,\beta}\})$, will only be used at the next transition (Barros 1998). Line 3 of the definition defines the state of the added components.

EXAMPLE. Consider the network of Figure 1. This model represents a buffered server, in its initial structure, when its composed of the buffer $Q$ and the single server $A$. When the number of clients in the queue is very high the executive receives an order to hire a new server. This server can be fired when clients decrease below a certain limit. The buffered server $\mathcal{S}$ is defined by

$$\mathcal{S} = (X_\mathcal{S},Y_\mathcal{S},\chi,M_\chi)$$

where

$Y_\mathcal{S} = J^*$

$J = \{c_0,c_1,\ldots,c_j,\ldots\}$ is a set of clients

$J^*$ is a sequence of jobs

$X_\mathcal{S} = J^* \times \{\textbf{change},\text{null}\} - \{(< >,\text{null})\}$

$M_\chi = (X_\chi,s_{0,\chi},S_\chi,\delta_\chi,\tau_\chi)$

with

$X_\chi = \{\textbf{change}\}$

$S_\chi = \{s_{0,\chi},s_{1,\chi}\}$

$\tau_\chi(s_{0,\chi}) = \tau_\chi(s_{1,\chi}) = \infty$

$\delta_\chi(s_{0,\chi},e,\textbf{change}) = s_{1,\chi}$

$\delta_\chi(s_{1,\chi},e,\textbf{change}) = s_{0,\chi}$

$\gamma(s_{0,\chi}) = (D_0,\{M_{i,0}\},\{I_{i,0}\},\{Z_{i,0}\})$

$\gamma(s_{1,\chi}) = (D_1,\{M_{i,1}\},\{I_{i,1}\},\{Z_{i,1}\})$

where

$D_0 = \{Q,A\}$
$D_1 = \{Q,A,B\}$
$M_{Q,0} = M_{Q,1} = (X_Q,s_{0,Q},S_Q,Y_Q,\delta_Q,\lambda_Q,\tau_Q)$
$M_{A,0} = M_{A,1} = (X_A,s_{0,A},S_A,Y_A,\delta_A,\lambda_A,\tau_A)$
$M_{B,1} = (X_B,s_{0,B},S_B,Y_B,\delta_B,\lambda_B,\tau_B)$
$I_{A,0} = I_{A,1} = I_A = \{Q\}$
$I_{\chi,0} = I_{\chi,1} = I_\chi = \{\mathcal{S}\}$
$I_{Q,0} = \{\mathcal{S},A\},\ I_{\mathcal{S},0} = \{A\}$
$Z_{Q,0}: X_{\mathcal{S}} \times X_A \rightarrow X_Q$
$Z_{A,0} = Z_{A,1} = Z_A$, and $Z_A: X_Q \rightarrow X_A$
$Z_{\chi,0} = Z_{\chi,1} = Z_\chi$, and $Z_\chi: X_{\mathcal{S}} \rightarrow X_\chi$
$Z_{\mathcal{S},0}: Y_A \rightarrow Y_{\mathcal{S}}$
$I_{B,1} = \{Q\}, I_{\mathcal{S},1} = \{A,B\}$,
$I_{Q,1} = \{\mathcal{S},A,B\}$
$Z_{Q,1}: X_{\mathcal{S}} \times X_A \times X_B \rightarrow X_Q$
$Z_B: X_Q \rightarrow X_B$
$Z_{\mathcal{S},1}: Y_A \times Y_B \rightarrow Y_{\mathcal{S}}$

The network initial structure is associated with partial state $s_{0,\chi}$, and is represented in Figure 1.

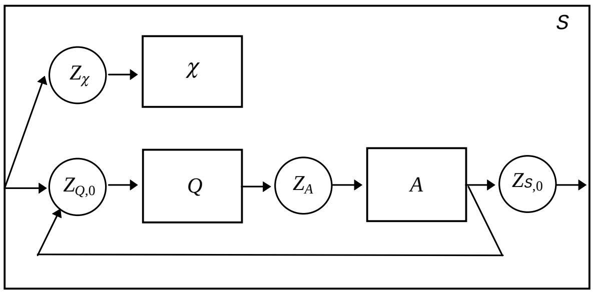

Figure 1: One Server Network

When the executive receives an order to hire a new server it changes to state $s_{1,\chi}$, where the associated structure is represented in Figure 2. The network can return to its initial structure after firing server *B*.

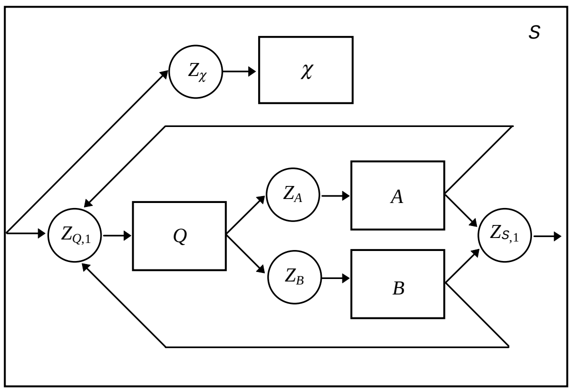

Figure 2: Two Server Network

## 5 THE ABSTRACT SIMULATORS

In this section, we describe the abstract simulators necessary to perform the operations implicit in the DSDE models. These simulators allow a description of models independently of the actual simulation procedure, and thus encourages model reuse. The *Simulator* is able to perform simulations using the implicit behavior contained in basic models, and the *NetworkSimulator* can simulate dynamic structure network models. The *Synchronizer* manages the simulation time. Separation between models and the simulation procedure was introduced by Zeigler (1984).

### 5.1 The Synchronizer

The *Synchronizer* coordinates the overall simulation. Its *child* is the topmost NetworkSimulator (or a Simulator if there is only an atomic model). The simulation starts when the Synchronizer receives the (START,*t*) message represented in Figure 3, and stops when its *child* is passive ($t_{N,child} = \infty$).

```
When receive (START,$t$)
  send (START,0) to child
  while ($t_{N,child} \neq \infty$) do
    "Computes the output of all the models"
    send (OUTPUT,$t_{N,child}$) to child
    "Makes the transition"
    send (TRANSITION,$t_{N,child}$,$\phi$) to child
  endWhile
end
```

Figure 3: Synchronizer Start Message

The message (OUTPUT,$t_{N,child}$) ensures that every model keeps a copy of its output just before it changes state. Thus, transitions can be executed in any order. The message (TRANSITION,$t_{N,child}$,$\phi$) is sent to the child and changes its state.

### 5.2 The NetworkSimulator

*NetworkSimulators* are attached to network models. The start message, described in Figure 4, is sent to every child, $t_N$ (time of next event) is set to the minimum of $t_{N,I}$ of children, and $t_L$ (time of last event) is set to the current time *t*. This message is sent just before the simulation begins and sets the component in its initial state.

```
When receive (START,$t$)
    send (START,$t$) to $\{I \mid I \in C\}$
    $t_L \leftarrow t$
    $t_N \leftarrow \min\{t_{N,I} \mid I \in C\}$
end
```

Figure 4: Network Simulator Start Message

We define $v_I$ as the value received from component *i*. This value can be the component output, if the component is not the network *N*, or is the network input otherwise.

$$v_I = \begin{cases} Y_I & \text{if } I \neq N \\ x & \text{if } I = N \end{cases}$$

The simulator transition is handled by the (TRANSITION,*t*,*x*) method shown in Figure 5, where *D* represents the current set of network components and *C* represents the set *D* plus the executive. If this message is sent to the executive, the structure of the network can change. To guarantee a correct initialization of the added components, the (START,*t*) message is sent to the new components after the executive transition.

```
When receive (TRANSITION,$t$,$x$)
1:  if $t \notin [t_L,t_N]$ then ERROR endIf
2:  if $t < t_N$ and $x = \phi$ then RETURN endIf
3:  $D' \leftarrow D$
4:  send (TRANSITION,$t$,$Z_I(\bigtimes_{j\in I_I} v_j)$) to $\{I \mid I \in D\}$
5:  send (TRANSITION,$t$,$Z_\chi(\bigtimes_{j\in I_\chi} v_j)$) to $\chi$
6:  send (START,$t$) to $\{I \mid I \in D - D'\}$
7:  $t_L \leftarrow t$
8:  $t_N \leftarrow \min\{t_{N,I} \mid I \in C\}$
end
```

Figure 5: Network Simulator Transition

Line 1 checks for state validity. In the definition of the total state set *Q*, the elapsed time *e* is constrained to $0 \leq e \leq \tau(s)$. Applying the equality $e = t - t_L$, this constraint can be written as $t_L \leq t \leq t_N$, that is used to identify errors. Line 2 ignores the transition message if the input is null and component is not schedule to change. Line 3 stores the current set of components on variable $D'$. Line 4 computes the transition in all the components excluding the executive. Thus changes in structure will only affect the next transition as required by the formalism. Line 5 computes the executive transition and the new network structure. Line 6 sends the START message to the new components. Lines 7 and 8 set the time of the last event and the time of the next event.

The network output function is defined in Figure 6. This function is called before each transition for correct formalism interpretation, ensuring that the output of a component does not depend on the transition order.

```
When receive (OUTPUT,$t$)
   if $t = t_N$ then
    send (OUTPUT,$t$) to $\{I \mid I \in C\}$
    $y \leftarrow Z_N(\bigtimes_{I\in I_N} y_I)$
   else
    $y \leftarrow \phi$
   endIf
end
```

Figure 6: NetworkSimulator Output Function

The output of a component *I* can be accessed by variable $y_i$.

## 5.3 The Simulator

The *Simulator* is attached to basic models. The message (START,*t*), represented in Figure 7, is called when a new model is placed in simulation at time *t*. It guarantees the updating of the equivalent total set when models are added, by the initialization of $t_L$ and $t_N$ variables. The start message is also executed when the simulation starts and initializes all the models.

```
when receive (START,$t$)
    $t_L \leftarrow t$
    $s \leftarrow s_0$
    $t_N \leftarrow t + \tau(s)$
end
```

Figure 7: Simulator Start Message

When the simulator receives the (TRANSITION,*x*,*t*) message it performs the model transition. Null values sent to a simulator are ignored. Simulator external transition is described in Figure 8. The time elapsed since the last transition of element *i* at time *t* is computed by $e = t - t_L$.

```
when receive (TRANSITION,$t$,$x$)
    if $t \notin [t_L,t_N]$ ERROR end
    if $t < t_N$ and $x = \phi$ then RETURN endIf
    $s \leftarrow \delta(s,t - t_L,x)$
    $t_L \leftarrow t$
    $t_N \leftarrow t + \tau(s)$
end
```

Figure 8: Simulator Transition

The output function of an atomic component is represented in Figure 9. This value must be stored before the change of state caused by the transition function.

```
when receive (OUTPUT,$t$)
    if $t = t_N$ then
        $y \leftarrow \lambda(s)$
    else
        $y \leftarrow \phi$
    endIf
end
```

Figure 9: Simulator Output Function

Because the (OUTPUT,t) message implements model output function, its value is null if time $t \neq t_N$.

## 6 CONCLUSIONS

We have presented the DSDE formalism and we have show that the DSDE formalism is closed under coupling, what permits to build models is a hierarchical and modular manner. The abstract simulators necessary to run DSDE models were also described. Abstract simulators permit to describe models independently from the actual simulation procedure, and are of particular importance to obtain model reuse.

## REFERENCES

Barros, F. J. 1995. Dynamic Structure Discrete Event System Specification: A New Formalism for Dynamic Structure Modeling and Simulation. In *Proceedings of the 1995 Winter Simulation Conference,* 781-785. Piscataway, New Jersey: IEEE Press.

Barros, F. J. 1996a. Dynamic Structure Discrete Event System Specification: Formalism, Abstract Simulators and Applications. *Transactions of the Society for Computer Simulation* 13(1): 35-46.

Barros, F. J. 1996b. Dynamic Structure Discrete Event System Specification: Structural Inheritance in the DELTA Environment. *Proceedings of the Sixth Annual Conference on AI, Simulation and Planning in High Autonomy Systems*, 141-147.

Barros, F. J. 1997a. Dynamic Structure Modeling and Simulation of the Eratosthenes Sieve for Prime Numbers. In *Proceedings of the 30th Annual Simulation Symposium*, 184-189. Piscataway, New Jersey: IEEE Press.

Barros, F. J. 1997b. Dynamic Structure Discrete Event Systems: A Comparison of Methodologies and Environments. In *Proceedings of SPIE 11th Annual International Symposium on Aerospace/Defense Sensing, Simulation and Controls: Enabling Technology for Simulation Science*, 3083: 268-277.

Barros, F. J. 1997c. Modeling Formalisms for Dynamic Structure Systems. *ACM Transactions on Modeling and Computer Simulation* 7(4): 501-515.

Barros, F. J. 1998. Handling Simultaneous Events in Dynamic Structure Models. *Proceedings of SPIE 12th Annual International Symposium on Aerospace/ Defense Sensing, Simulation and Controls: Enabling Technology for Simulation Science,* 3083. To be published.

Chow, A. C. 1994. Parallel DEVS: A Parallel, Hierarchical, Modular Modeling Formalism and its Distributed Simulator. *SCS Transactions* 13(2): 55-67.

Zeigler, B. P. 1976. *Theory of Modelling and Simulation*. New York: Wiley.

Zeigler, B. P. 1984. *Multifaceted Modelling and Discrete Event Simulation*. London: Academic Press.

Zeigler, B. P. 1990. *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*. New York: Academic Press.

## AUTHOR BIOGRAPHY

**FERNANDO J. BARROS** is an assistant professor at the Department of Informatics of the University of Coimbra. He received M.S. and Ph.D. degrees from the University of Coimbra. He is a member of SCS and ACM. His research interests include Theory of Modeling and Simulation, and adaptive systems.