CHAPTER

BASIC FORMALISMS: COUPLED MULTI-COMPONENT SYSTEMS

7

CONTENTS

7.1 Discrete Event Specified Network Formalism 168
7.1.1 Classic DEVS Coupled Models 168
Closure Under Coupling of Classic DEVS 170
7.1.2 Parallel DEVS Coupled Models 171
7.1.3 Closure Under Coupling of Parallel DEVS 171
Output Function 172
Internal Transition Function 172
External Transition Function 172
7.1.4 The Confluent Transition Function 173
7.2 Multi-Component Discrete Event System Formalism 174
7.2.1 Cellular Automata Multi-Component DEVS of GOL Event Model 176
Cellular Automata Multi-Component DEVS of GOL Event Model 177
7.2.2 Event Scheduling Models 178
Implementing Event Scheduling Simulation Systems in Imperative Programming Languages ... 178
7.2.3 Combined Event Scheduling, Activity Scanning Simulation Strategy 179
7.2.4 Process Interaction Models 180
7.2.5 Translating Non-Modular Multi-Component DEVS Models Into Modular Form 181
7.2.6 State Updating in Distributed Simulation 182
7.3 Discrete Time Specified Network Formalism 183
Delay-Free (Algebraic) Cycles 183
Definition of Discrete Time Coupled Models 184
Closure Under Coupling of DTSS 184
7.4 Multi-Component Discrete Time System Formalism 185
Spatial DTSS: Cellular Automata 186
7.5 Differential Equation Specified Network Formalism 187
Closure Under Coupling of DESS 188
7.6 Multi-Component Differential Equations Specified System Formalism 188
7.6.1 Spatial DESS: Partial Differential Equation Models 189
7.7 Multi-Component Parallel Discrete Event System Formalism 191
7.8 Summary 191
7.9 Sources 192
Appendix 7.A 192
References 193

In Sections 5.7 and 5.8 we introduced the concept of system specification at the multi-component and network of systems (also called coupled systems) level. Both levels allow us to model systems by composing smaller systems together. In the multi-component system specification, composition is done non-modularly in that one component's state transitions can directly change another component's state. In contrast, the network of systems specification provides a means to couple standalone systems by connecting their output and input interfaces.

In Chapter 6 we introduced system specification formalisms as shorthand means to specify systems. A formalism defines a set of background conventions as features which are common to all systems of the class it specifies. We identified DESS, DTSS, and DEVS as the three fundamental system formalisms for simulation modeling.

The formalisms of Chapter 6 were at the I/O systems level or structured systems levels. In this chapter, we formulate the basic system formalisms at the *multi-component* and the *network of systems* (coupled system) level. Further, for each formalism, we will provide conditions that guarantee that a coupling of systems in this formalism defines a basic system in the same formalism – we say that the formalism is *closed under coupling*. Closure under coupling allows us to use networks of systems as components in a larger coupled systems, leading to hierarchical, modular construction.

Non-modular system specification is important because many traditional forms of modeling (e.g., the classical world views of discrete event simulation) are non-modular. However, the advantages of modular paradigms are decisive in the age of distributed simulation and model repositories. Thus, translation from non-modular to modular specifications is an important topic of this chapter.

7.1 DISCRETE EVENT SPECIFIED NETWORK FORMALISM

The Discrete Event Specified Network formalism (DEVN), also called the DEVS coupled model specification provides a basis for modular construction of discrete event models. Such modular construction, although, at the time of this writing, is not as common an approach as the non-modular alternative, is destined to be more and more dominant. The reason is that modularity is more natural for distributed simulation and supports reuse through model repositories. Interoperability through distributed simulation with repository reuse is the goal of the High Level Architecture (HLA) standard being promulgated of the U.S. Department of Defense.

In discrete event coupled networks, components are DEVS systems coupled exclusively through their input and output interfaces. Components do not have the possibility to access and influence the states or the timing of other components directly. All interactions have to be done by exchanging messages. More specifically, the events generated by one component at its output ports are transmitted along the couplings to input ports where they cause external events and state transitions at the influenced components.

Besides treating classical DEVS in the following discussion, we also will consider the revision called Parallel DEVS. We will discuss the closure under coupling of both classic and Parallel DEVS.

7.1.1 CLASSIC DEVS COUPLED MODELS

Recall the definition of coupled DEVS models in Chapter 4. Instead of repeating this definition here, we provide the more abstract version which does not explicitly work with input/output ports but corresponds to the abstract definition of network of systems of Chapter 5. The translation from structured to abstract version follows the lines discussed in Chapter 5. The definition for DEVS coupled models adopted here differs only slightly from the general definition in Chapter 5 as the interface map Z_d for a component d is broken up into several interface mappings $Z_{i,d}$, one for each influencing component i. In addition to the general definition of coupled models, classic DEVS coupled models employ a Select function which is used for tie-breaking in case of equal next event times in more than one component. The structure of the DEVN or *coupled* DEVS *model* is

$$N = \left\langle X, Y, D, \{M_d\}, I_d, Z_{i,d} Select \right\rangle$$

with

$$\begin{aligned}&X \text{ a set of input events,}\\ &Y \text{ a set of output events, and}\\ &D \text{ a set of component references.}\end{aligned}$$

For each $d \in D$,

$$M_d \text{ is a classic DEVS model.}$$

For each $d \in D \cup \{N\}$,

$$I_d \text{ is the influencer set of d: } I_d \subseteq D \cup \{N\}, d \notin I_d$$

and for each $i \in I_d$

$$\begin{aligned}&Z_{i,d} \text{ is a function, the i-to-d output translation with}\\ &\quad Z_{i,d} : X \to X_d, \text{ if } i = N.\\ &\quad Z_{i,d} : Yi \to Y, \text{ if } d = N.\\ &\quad Z_{i,d} : Yi \to X_d, \text{ if } d \neq N \text{ and } i \neq N.\end{aligned}$$

Note that we use the influencer sets to specify the influence propagation in order to be consistent with the approach of Chapter 5. Earlier definitions, in the first edition of TMS and elsewhere, employed the influencee sets, since as we see below, this is more natural for classic DEVS where only one imminent component is activated. However, either family of sets gives the same information and its more convenient to use the influencer sets for Parallel DEVS.

Exercise 7.1. Show that given the family of influencee sets we can derive the family of influencer sets, and conversely.

Finally, Select is a function

$$\text{Select} : 2^D \to D$$

with Select$(E) \in E$. Select is the tie-breaking function to arbitrate the occurrence of simultaneous events (see below).

Recall (Chapter 5) that the presence of N as a potential source or destination component is a device that makes it easy to represent external input and external output couplings.

CLOSURE UNDER COUPLING OF CLASSIC DEVS

Given a coupled model, N, with components that are classic DEVS, we associate with it a basic DEVS, called the *resultant*:

$$DEVSN = \langle X, Y, S, \delta_{\text{ext}}, \delta_{\text{int}}, \lambda, \text{ta} \rangle$$

where $S = \times_{d \in D} Q_d$, $\text{ta} : S \rightarrow \mathbb{R}_0^+ \cup \{\infty\}$ is defined by

$$\text{ta}(s) = minimum\{\sigma_d | d \in D\}$$

where for each component $d \sigma_d = \text{ta}_d(s_d) - -e_d$, i.e., σ_d is the time remaining to the next event in component d.

Let the set of $imminents$, $IMM(s) = \{d | d \in D \wedge \sigma_d = \text{ta}(s)\}$. $Imminents$ is the set of components that have minimum remaining time σ_d, i.e., they are candidates for the next internal transition to occur.

Let $d* = \text{Select}(IMM(s))$. This is the component whose output and internal transition functions will be executed because it was selected using the tie breaking function Select.

Now, we define $\delta_{\text{int}} : S \rightarrow S$. Let $s = (..., (s_d, e_d)...)$. Then $\delta_{\text{int}}(s) = s' = (..., (s'_d, e'_d)...)$, where

$$(s'_d, e'_d) = \begin{cases} (\delta_{\text{int}}, d(s_d), 0) & \text{if } d = d* \\ (\delta_{\text{ext}}, d((s_d, e_d + \text{ta}(s)), x_d), 0) & \text{if } d* \in I_d \wedge x_d \neq \emptyset \\ (s_d, e_d + \text{ta}(s)) & \text{otherwise} \end{cases}$$

where $x_d = Z_{d*,d}(\lambda_{d*}(s_{d*}))$. That means, the resultant's internal transition function changes the state of the selected imminent component $d*$ according to its $(d *' s)$ internal transition function and updates all influenced components according to the inputs produced by the output of $d*$. In all other components, d, only the elapsed times, e_d, are updated.

We define $\delta_{\text{ext}} : Q \times X \rightarrow S$, $\delta_{\text{ext}}((s, e), x) = s' = (..., (s'_d, e'_d)...)$ by

$$(sd', ed') = \begin{cases} (\delta_{\text{ext}}, d((s_d, e_d + e), x_d), 0) & \text{if } N \in I_d \wedge x_d \neq \emptyset \\ (sd, ed + e) & \text{otherwise} \end{cases}$$

where $x_d = Z_{N,d}(x)$. That is, all components influenced by the external input x change state according to the input x_d transmitted to them as converted by their respective interface mappings. All other components increment their elapsed times by e.

Finally, we define the resultant's output function, $\lambda : S \rightarrow Y$:

$$\lambda(s) = \begin{cases} Z_{d*,N}(\lambda_{d*}(s_{d*})) & \text{if } d* \in I_N \\ \emptyset & \text{otherwise.} \end{cases}$$

This says that we translate the output of the selected imminent component, $d*$, to an output of the coupled model through interface map $Z_{d*,N}$. This, of course, is only for the case that $d*$ actually sends external output.

7.1.2 PARALLEL DEVS COUPLED MODELS

The structure of a *coupled DEVS model* for Parallel DEVS is almost identical to that of the classical DEVS except for the absence of the Select function:

$$N = \langle X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}\rangle$$

with

X a set of input events,
Y a set of output events, and
D a set of component references.

For each $dinD$,

M_d is a *Parallel DEVS* model.

For each $d \in D \cup \{N\}$, I_d is the influencers set of d and $Z_{i,d}$ is the i-to-d output translation with the same definition as in classical DEVS (see above).

7.1.3 CLOSURE UNDER COUPLING OF PARALLEL DEVS

We consider a coupled model

$$N = \langle X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}\rangle$$

where each component, M_d is a Parallel DEVS.

We demonstrate closure under coupling by constructing the resultant of the coupled model and showing it to be a well-defined *Parallel DEVS*. The resultant is (potentially) a *Parallel DEVS*

$$DEVS_N = \langle X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \delta_{\text{con}}, \lambda, \text{ta}\rangle\,.$$

As with classic DEVS, the state set is a crossproduct, $S = \times_{d \in D} Q_d$. Likewise, the time advance function is defined by

$$\text{ta}(s) = minimum\{\sigma_d / d \in D\},\ \text{where } s \in S \text{ and } \sigma_d = \text{ta}(s_d) - e_d$$

The major difference with classic DEVS comes in the definitions of the remaining functions. To make these definitions, we partition the components into four sets at any transition. Let $s = (..., (s_d, e_d), ...)$. Then $IMM(s)$ are the imminent components whose outputs will be generated just before the next transition. Since there is no Select function, all of the imminents will be activated. $INT(s)$ is the subset of the imminents that have no input messages. $EXT(s)$ contains the components receiving input events but not scheduled for an internal transition. CONF(s) contains the components receiving input events and also scheduled for internal transitions at the same time. $UN(s)$ contains the remaining components. Formally,

$IMM(s) = \{d / \sigma_d = \text{ta}(s)\}$ (the imminent components),

$INF(s) = \{d | i \in I_d, i \in IMM(s) \wedge x_d^b \neq \Phi\}$ (components about to receive inputs),
where $x_d = \{Z_{i,d}(\lambda_i(s_i)) | i \in IMM(s) \cap I_d\}$
$CONF(s) = IMM(s) \cap INF(s)$ (confluent components),
$INT(s) = IMM(s) - INF(s)$ (imminent components receiving no input),
$EXT(s) = INF(s) - IMM(s)$ (components receiving input but not imminent),
$UN(s) = D - IMM(s) - INF(s)$.

OUTPUT FUNCTION

The output is obtained by collecting all the external outputs of the imminents in a bag:

$$\lambda(s) = \{Z_{d,N}(\lambda_d(s_d))/d \in IMM(s) \wedge d \in I_N\}.$$

INTERNAL TRANSITION FUNCTION

The resultant internal transition comprises four kinds of component transitions: internal transitions of $INT(s)$ components, external transitions of $EXT(s)$ components, confluent transitions of $CONF(s)$ components, and the remainder, $UN(s)$, whose elapsed times are merely incremented by ta(s). (The participation of $UN(s)$ can be removed in simulation by using anabsolute time base rather than the relative elapsed time.) Note that, by assumption, this is an internal transition of the resultant model, and there is no external event being received by the coupled model at this time.

We define

$$\delta_{\text{int}}(s) = (..., (s'_d, e'_d), ...)$$

where

$$(s'_d, e'_d) = \begin{cases} (\delta_{\text{int}}, d(s_d), 0) & \text{for } d \in INT(s), \\ (s'_d, e'_d) = (\delta_{\text{ext}}, d(s_d, e_d + \text{ta}(s), x_d^b), 0) & \text{for } d \in EXT(s), \\ (s'_d, e'_d) = (\delta_{\text{con}}, d(s_d, x_d^b), 0) & \text{for } d \in CONF(s), \\ (s'_d, e'_d) = (s_d, e_d + \text{ta}(s)) & \text{otherwise} \end{cases}$$

where x_d^b is as defined above.

EXTERNAL TRANSITION FUNCTION

To construct δ_{ext} of the resultant, let

$$\delta_{\text{ext}}(s, e, x^b) = (..., (s'_d, e'_d), ...)$$

where $0 < e < \text{ta}(s)$ and

$$(s'_d, e'_d) = \begin{cases} (\delta_{\text{ext}}, d(s_d, e_d + e, x_d^b), 0) & \text{for } N \in I_d \wedge x_d^b \neq \Phi, \\ (s_d, e_d + e) & \text{otherwise,} \end{cases}$$

where

$$x_d^b = \{Z_{N,d}(x)/x \in x^b \wedge N \in I_d\}.$$

The incoming event bag, x^b is translated and routed to the event bag, x_d^b, of each influenced component, d. The resultant's external transition comprises all the external transitions of the influenced children. Note that by assumption, there are no internally scheduled events at this time ($e < \mathrm{ta}(s)$).

7.1.4 THE CONFLUENT TRANSITION FUNCTION

Finally, we construct the δ_{con} of the resultant. This is what happens in the coupled model when some of the components are scheduled to make internal transitions and are also about to receive 7 external events. Fortunately, it turns out that the difference between δ_{con} and the earlier defined, δ_{int}, is simply the extra confluent effect produced by the incoming event bag, x^b, at elapsed time $e = \mathrm{ta}(s)$. By redefining the influencee set $INF(s)$ as $INF'(s)$, which includes the additional influencees from the incoming couplings, I_N, we develop three similar groups for δ_{con}. Let

$$INF'(s) = \{d|(i \in I_d, i \in IMM(s) \vee N \in I_d) \wedge x_d^b \neq \Phi\}$$

where $x_d^b = \{Z_{i,d}(\lambda_i(s_i))|i \in IMM(s) \wedge i \in I_d\} \cup \{Z_{n,d}(x)|x \in x^b \wedge N \in I_d\}$ (this is a union of bags so common elements are retained. The first bag collects outputs from imminent components that are coupled to the receiving component, d. The second bag collects external inputs that are sent to the receiver through the external input coupling).

$$\begin{aligned} CONF'(s) &= \mathbf{IMM(s)} \cap INF'(s) \\ INT'(s) &= \mathbf{IMM(s)} - INF'(s), \\ EXT'(s) &= INF'(s) - \mathbf{IMM(s)}. \end{aligned}$$

We define

$$\delta_{\mathrm{con}}(s, x^b) = (\ldots, (s'_d, e'_d), \ldots),$$

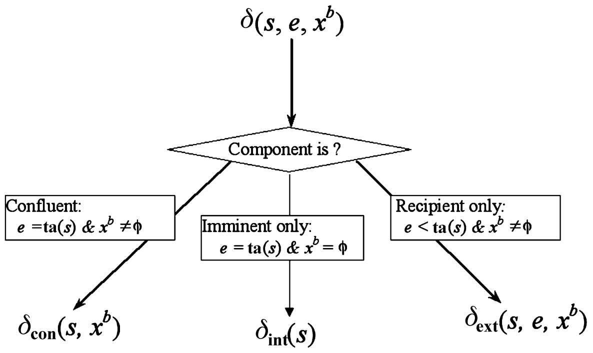
where

$$(s'_d, e'_d) = \begin{cases} (\delta_{\mathrm{int},d}(s_d), 0) & \text{for } d \in INT'(s), \\ (\delta_{\mathrm{ext},d}(s_d, e_d + \mathrm{ta}(s), x_d^b), 0) & \text{for } d \in EXT'(s), \\ (\delta_{\mathrm{con},d}(s_d, x_d), 0) & \text{for } d \in CONF'(s), \\ (s_d, e_d + \mathrm{ta}(s)) & \text{otherwise} \end{cases}$$

where x_d^b is defined above.

Hierarchical consistency is achieved here by the bag union operation that gathers all external events, whether internally or externally generated, at the same time into one single event group. By such consistency, we mean that a component would experience the same inputs independently of the manner in which we chose to decompose a model (see Chow, 1996 for more details).

From the definition of the $\delta_{\mathrm{int}'}$, $\delta_{\mathrm{con}'}$, and $\delta_{\mathrm{ext}'}$, we see that they are special cases of one encompassing transition function, $\delta(s, e, x^b)$ applied to each component. As shown in Fig. 7.1, δ_{int} is applied



δ(s, e, x^b)
Component is ?
Confluent: e = ta(s) & x^b ≠ φ
Imminent only: e = ta(s) & x^b = φ
Recipient only: e < ta(s) & x^b ≠ φ
δ_con(s, x^b)
δ_int(s)
δ_ext(s, e, x^b)

FIGURE 7.1

The overall transition function.

to imminent (non-confluent), i.e., when $(s, e, x^b) = (s, \text{ta}(s), \Phi)$; δ_{con} is applied to confluent components, i.e., $(s, e, x^b) = (s, \text{ta}(s), x^b)$ where $x_d^b \neq \Phi$; finally, δ_{ext} is applied to receiver (non-confluent) components, i.e., (s, e, x^b) where $0 < e < \text{ta}(s)$ and $x_d^b \neq \Phi$.

The overall transition function is the basis for implementation of coupled Parallel DEVS. In terms of the special case functions, it is defined by:

$$\begin{aligned}
&\delta(s, e, x^b) = \delta_{\text{ext}}(s, e, x^b) \text{ for } 0 < e < \text{ta}(s), \text{ and } x^b \neq \Phi, \\
&\delta(s, \text{ta}(s), x^b) = \delta_{\text{con}}(s, x^b) \text{ for } x^b \neq \Phi, \\
&\delta(s, \text{ta}(s), \Phi)) = \delta_{\text{int}}(s).
\end{aligned}$$

Exercise 7.2. Write the generic transition function

$$\delta(s, e, x^b) = (..., (s'_d, e'_d), ...)$$

for the resultant of a coupled model in terms of the generic functions of its components.

7.2 MULTI-COMPONENT DISCRETE EVENT SYSTEM FORMALISM

Recall (Chapter 5) that in non-modular coupling, components directly influence each other through their state transitions. In DEVS multi-component non-modular coupled models, events 9 occurring in one component may result in state changes and rescheduling of events in other components. The simulation strategies realized in many commercial simulation languages and systems, known as "world views", fall into the category of multi-component DEVS. In this section, we will make the connection from the system theoretical DEVS approach to these discrete event simulation strategies. However, before we introduce the formalism for multi-component DEVS and formulate the GOL event model discussed in Chapter 3 as a multi-component DEVS.

A multi-component DEVS is a structure

$$multiDEVS = \langle X, Y, D, \{M_d\}, Select \rangle$$

where

X, Y are the input and output event sets, D is the set of component references, and Select : $2^D \to D$ with Select$(E) \in E$ is a tie-breaking function employed to arbitrate in case of simultaneous events.

For each $d \in D$

$$M_d = \left\langle S_d, I_d, E_d, \delta_{\text{ext},d}, \delta_{\text{int},d}, \lambda_d, \text{ta}_d \right\rangle$$

where

S_d is the set of sequential states of d,
$Q_d = (s, e_d)|s \in S_d, e_d \in \mathbb{R}$ is the set of total states of d,
$I_d \subseteq D$ is the set of influencing components,
$E_d \subseteq D$ is the set of influenced components,
$\delta_{\text{ext},d} : \times_{i \in I_d} Q_i \times_{j \in E_d} X \to \times Q_j$ is the external state transition function,
$\delta_{\text{int},d} : \times_{i \in I_d} Q_e \to \times_{j \in E_d} Q_j$ is the internal state transition function,
$\lambda_d : \times_{i \in I_d} Q_i \to Y$ is the output event function, and
$\text{ta}_d : \times_{i \in I_d} Q_i \to \mathbb{R} + 0 \cup \infty$ is the time advance function.

A multiDEVS works in the following way: Internal events are scheduled individually by each component $d \in D$ by the individual time advance functions ta_d. When an event occurs in one of the components, it is executed and this results in state changes through the internal transition function $\delta_{\text{int},d}$ and eventually output events for the multiDEVS defined in the output function λ_d. Note, that the transition function depends on total states q_i of the influencing components I_d and change any total state q_j of the influencing components E_d, also including the elapsed time e_j. When events in different components are imminent, the Select function is used to arbitrate among them. External events at the multiDEVS's input interface can be handled by any of the external state transition functions $\delta_{\text{ext},d}$ of the components, d. However, a component may not react to external inputs if its external state transition function is not defined.

A multiDEVS defines an atomic DEVS as follows. Given a multiDEVS as defined above, we associate a basic

$$DEVS = \langle X, Y, S, \delta_{\text{ext}}, \delta_{\text{int}}, \lambda, \text{ta} \rangle$$

where $S = \times_{d \in D} Q_d$. We define

$$\text{ta}(s) = min\{\sigma_d | d \in D\}$$

with $\sigma_d = \text{ta}_d(..., q_i, ...) - e_d$ is the remaining time until the next scheduled event in d.

We define the next scheduled component $d* = \text{Select}(\{d | \sigma_d = \text{ta}(s)\}$ and

$$\delta_{\text{int}}((s_1, e_1), (s_2, e_2), ..., (s_n, e_n)) = ((s_1', e_1'), (s_2', e_2'), ..., (s_n', e_n'))$$

with

$$(s'_j, e'_j) = \begin{cases} (sj, ej + \text{ta}(s)) & \text{if } j \notin E_{d*} \\ (s'_j, e'_j) = \delta_{\text{int},d*}((..., (s_i, e_i + \text{ta}(s)), ...)).j & \text{if } j \in E_{d*} \end{cases}$$

with $i \in I_d*$, i.e. the internal state transition of the DEVS is defined by the state transition of the next scheduled component $d*$.

The output of the system is defined by the output event of $d*$

$$\lambda(s) = \lambda_{d*}(..., q_i, ...), i \in I_d*$$

The external transition function δ_{ext} upon occurrence of an input event x is defined by the crossproduct of the external state transition functions

$$\delta_{\text{ext},d}((..., q_i, ...), e, x)$$

of all components $d \in D$. We define the overall external transition function by

$$\delta_{\text{ext},d}(((s_1, e_1), (s_2, e_2), ..., (s_n, e_n)), e, x) = ((s'_1, e'_1), (s'_2, e'_2), ..., (s'_n, e'_n))$$

with

$$(s'_j, e'_j) = \begin{cases} (s_j, e_j + e) & \text{if } j \notin E_d \\ \delta_{\text{ext}}, d((..., (s_i, e_i + e), ...)).j & \text{if } j \in E_d. \end{cases}$$

7.2.1 CELLULAR AUTOMATA MULTI-COMPONENT DEVS OF GOL EVENT MODEL

We illustrate the general ideas of event coupling in multi-component systems by recalling the discrete event cell space model introduced in Chapter 3. Clearly such a model is a multi-component DEVS model with identical components, uniform arrangement and couplings. The time advance function of a cell takes care of scheduling its birth and death based on its current fitness and the sum of its neighbors' states. (Note, that the time advance is infinity in case that no event is scheduled to happen.) A state event in one cell can influence the time advance of its influencees, possibly causing them to be rescheduled. We will see in the next chapter how the simulation algorithm takes care to plan the events in time, in particular, how events in influenced components $E_{i,j}$ are rescheduled upon changes in a neighboring cell.

For the state set of each cell $M_{i,j}$ we consider the set of pairs,

$$S_{i,j} = \{(state, fitness) | state \in 0, 1, fitness \in ?\}$$

with variables $state$ to denote the actual dead or alive state of the cell and the $fitness$ parameter representing the current fitness of the cell. Additionally we employ a help variable sum representing the sum of the alive neighbors

$$sum = |\{(k,l)|(k,l) \in I_{i,j}, s_{k,l}.state = 1\}|,$$

and an auxiliary variable $factor$ representing the increase or decrease of the fitness parameter

$$factor = \begin{cases} 1 & \text{if } sum = 3 \\ -2 & \text{if } q_{i,j}.state = 1 \wedge (sum < 2 | sum > 3) \\ 0 & \text{otherwise} \end{cases}$$

CELLULAR AUTOMATA MULTI-COMPONENT DEVS OF GOL EVENT MODEL

We illustrate the general ideas of event coupling in multi-component systems by recalling the discrete event cell space model introduced in Chapter 3. Clearly such a model is a multi-component DEVS model with identical components, uniform arrangement and couplings. The time advance function of a cell takes care of scheduling its birth and death based on its current fitness and the sum of its neighbors' states. (Note, that the time advance is infinity in case that no event is scheduled to happen.) A state event in one cell can influence the time advance of its influencees, possibly causing them to be rescheduled. We will see in the next chapter how the simulation algorithm takes care to plan the events in time, in particular, how events in influenced components $E_{i,j}$ are rescheduled upon changes in a neighboring cell.

For the state set of each cell $M_{i,j}$ we consider the set of pairs,

$$S_{i,j} = \{(state, fitness)|state \in \{0,1\}, fitness \in\}$$

with variables $state$ to denote the actual dead or alive state of the cell and the $fitness$ parameter representing the current fitness of the cell. Additionally we employ a help variable sum representing the sum of the alive neighbors

$$sum = |\{(k,l)|(k,l) \in I_{i,j}, s_{k,l}.state = 1\}|,$$

and an auxiliary variable $factor$ representing the increase or decrease of the fitness parameter

$$factor = \begin{cases} 1 & \text{if } sum = 3 \\ -2 & \text{if } q_{i,j}.state = 1 \wedge (sum < 2 | sum > 3). \\ 0 & \text{otherwise} \end{cases}$$

Given the current $fitness$, and based on the value of $factor$, we easily can compute the time to the next zero crossing and hence the time advance. Note that only in the dead state, with a positive $factor$, and in the alive state with negative $factor$, are events scheduled. Formally,

$$\text{ta}_{i,j}(q_{i,j}, q_{i,j+1}, q_{i+1,j}, q_{i,j-1}, q_{i-1,j}, q_{i+1,j+1}, q_{i+1,j-1}, q_{i-1,j+1}, q_{i-1,j-1}) =$$
$$= \begin{cases} \dfrac{-fitness}{factor} & \text{if } ((state = 1 \wedge factor < 0) | (state = 0 \wedge factor > 0)) \\ \infty & \text{otherwise} \end{cases}$$

Finally, the state transition function has to consider time scheduled events (birth or death) as well as changes in the environment. These can be specified as given in the following

$$\begin{aligned}
&\delta_{i,j}(q_{i,j}, q_{i,j+1}, q_{i+1,j}, q_{i,j-1}, q_{i-1,j}, q_{i+1,j+1}, q_{i+1,j-1}, q_{i-1,j+1}, q_{i-1,j-1}) = \\
&fitness = max\{6, fitness + factor \cdot e\} \\
&e_{i,j} = 0 \\
&\text{if } (state = 0 \wedge fitness = 0 \wedge factor \geq 0) \text{ then } state = 1 \\
&\text{else if } (state = 1 \wedge fitness = 0 \wedge factor \leq 0) \text{ then } state = 0,\ fitness = -2
\end{aligned}$$

where $factor$ is defined as above. First the $fitness$ parameter for the current time is computed and the elapsed time of the component is set to 0. This applies to external changes as well as for internal events. Now, the increase in $fitness$ at any event is equal to $factor \cdot e$. (Since the last update of the $fitness$ value occurred at the last event and e is the time since the last event and the factor value did not change since the last event (why?).) Thus, we check if an internal event is to occur ($fitness = 0$) and execute a birth or death, respectively. We switch the state value from alive (1) to death (0) or vice versa and reinitialize the fitness with -2 in case of a death.

7.2.2 EVENT SCHEDULING MODELS

We come back now to discuss the world views of discrete event simulation, first considered in Chapter 3, in the context of multi-component DEVS. There are three common types of discrete event simulation strategies employed in discrete event simulation languages and packages – the *event scheduling*, the *activity scanning*, and the *process interaction* world view, the latter being a combination of the first two. A strategy makes certain forms of model description more naturally expressible than others. We will model the "pure" forms of these strategies, that is, the forms in which only the features inherent to the basic conception are employed.

Event oriented models work with prescheduling of all events and there is no provision for activating events by tests on the global state. Event scheduling strategy can be captured by the multi-component DEVS as introduced above.

IMPLEMENTING EVENT SCHEDULING SIMULATION SYSTEMS IN IMPERATIVE PROGRAMMING LANGUAGES

Due its simplicity, event scheduling simulation is the preferred strategy when implementing customized simulation systems in procedural programming languages. One takes the approach that the components' states are realized by arbitrary data records and the state transition functions by several procedures which operate on these data. Most important, each active component defines a sort of "control state" similar to the DEVS's phase variable to denote different events types it has to process. For each such event type, one procedure is coded which implements the state transition for the event type. We can say that the set of event types $S_d^{control} = \{ev_1, ev_2, ev_3, ...ev_n\}$ divides the state transition function δ_d of component d into n functions δ_d^{evi}, each describing the activity of the component d when started in one of its control states, ev_1. At an event the component branches to one of its event routines depending on the control state (Algorithm 1). On such a model implementation an event list simulation algorithm can be realized which sorts components into the event list depending on their event times and then takes them out in sequence to process.

```
Component d
$S_{control, d}$ = \{ ev_1 , ev_2 , ..., ev_n \}
$\delta_{ss}$ (($s_{control, d}$ , ... ))
  case $s_{control, d}$
    $ev_1$ : call event-routine_1
    $ev_2$ : call event-routine_2
    ....
    $ev_n$ : call event-routine_n
```

7.2.3 COMBINED EVENT SCHEDULING, ACTIVITY SCANNING SIMULATION STRATEGY

We now augment the event oriented scheme to allow for activating components by means of contingency tests. Referring to Fig. 7.2, we can understand the additional model description power attained by these means. Suppose that at time t component d is state s_d with a elapsed time e_d of 0 (meaning that d just entered state s_d with an event at time t). Then a time of next event is scheduled at time $t + \text{ta}_d((...., q_i, ...))$. Unless d is rescheduled by the action of another imminent component, it will be activated at this time to undertake an internal transition. In the pure event scheduling strategy this will in either case result in a change of state in d and in a redefinition of its next activation time.

In contrast, in the activity scanning approach, events can be conditioned on a contingency test. When its time has expired and if the contingency test evaluates to true, the event will occur and the state will change. Otherwise, the event is said to be *due*. This means that the elapsed time component e_d is allowed to become greater than the time advance, so the total state set of a component d is $Q_d = \{(s_d, e_d)|s_d \in S_d, e_d \geq 0\}$. Once it becomes due, an event must be checked at any global change of state to see if its activation condition has become true. Whenever this condition becomes true, the event can be activated.

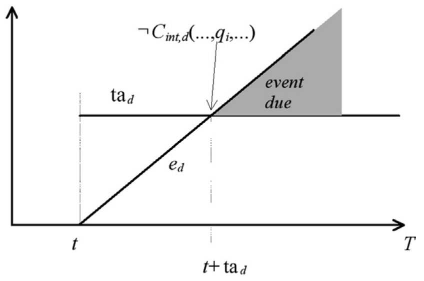


FIGURE 7.2

The meaning of due events.

To capture the activity scanning strategy, we introduce so-called *activity scanning multi-component* DEVS as follows:

$$ASDEVS = \langle X, Y, D, \text{Select} \rangle$$

with the same interpretation as for basic multi-component DEVS. However, the components $d \in D$ are

$$M_d = \left\langle S_d, E_d, I_d, \delta_{\text{ext},d}, C_{int,d}, \delta_{\text{int},d}, \lambda_d, \text{ta}_d \right\rangle$$

with Sd, Ed, I_d, $\delta_{\text{ext},d}$, $\delta_{\text{int},d}$, λ_d, ta_d the same as in the basic multi-component DEVS. Additionally, an activity DEVS employs an *event condition function* $Cd : \times_{i \in I_d} Q_i \to Bool$.

An activity scanning DEVS works in the following way: Each component has its own state transitions whose occurrence is conditioned by the event condition function. Whenever the time elapses and the event condition becomes true, the component is ready to be activated. The Select-function is used to arbitrate between components having true event conditions simultaneously. Formally, with an $ASDEVS = (X_N, Y_N, D, \text{Select})$ we associate the following DEVS

$$DEVS = \langle X, Y, S, \delta_{\text{ext}}, \delta_{\text{int}}, \lambda, \text{ta} \rangle$$

with $S = \times_{d \in D} Q_d$. In the definition of the time advance function, let $\sigma_d = \text{ta}_d(..., q_i, ...) - e_d$ be the time left to the next time in d. We distinguish two cases. For case (1) if $\exists d \in D : \sigma_d \leq 0 \wedge C_{int,d}(..., q_i, ...)$, i.e. at current simulation time t there is a component which is ready to be activated, then we define $\text{ta}(s) = 0$. Otherwise we have case (2) and we have to advance the time to the time of the next activatable component, i.e., we define $\text{ta}(s) = min\{\sigma_d | d \in D \wedge \sigma_d > 0 \wedge C_{int,d} * (..., q_i, ...)\})$. In either case, we define $d* = Select(\{d | \sigma_d \leq \text{ta}(s) \wedge C_{int,d}(..., q_i, ...)\})$ as the next activatable component. The internal transition function $\delta_{\text{int}}(s)$, the output function $\lambda(s)$ and the external transition function $\delta_{\text{ext}}(s, e, x)$ are defined in the same way as for event scheduling DEVS.

7.2.4 PROCESS INTERACTION MODELS

Process interaction simulation strategy is basically a combined event scheduling – activity scanning procedure. The distinguishing feature is that a model component description can be implemented as a unit rather than being separated into a number of unconnected events and activity routines. The advantage is that the program structure maintains a closer relation to the model structure and by scanning the source code, the reader gets a better impression on the model's behavior.

Each component's dynamic behavior is specified by one (or also more) routine which describes the state changes during the component's life cycle as a sequential program. This sequential program can be interrupted in two ways:

1. by the specification of time advance
2. by a contingency test

Whenever such an interrupt occurs, the component pauses until the activation condition is satisfied, i.e., either the specified time advance has elapsed or the contingency test has become true. Hence, an important state variable in a process is its *program counter* which defines its activation point and, therefore, the next state transition to be executed. The program counter represents a control state in each component. It usually shows the particular phase of processing that a component is in. Thus, it is comparable to the phase variable employed in atomic DEVS models.

The process interaction world view further breaks down into two views. Each view corresponds to a different assumption as to what are the active and passive components in building models of manufacturing or other processing systems. In the first view, the active components are taken to be the entities that do the processing, e.g., the machines, servers, and so on. In the second approach, the active components are the flowing elements, that is the customers, workpieces, packets, etc. Whereas the first is regarded to be the prototypical process interaction world view, the second is regarded as a variation of the first and often referred to as the *transaction world view*.

7.2.5 TRANSLATING NON-MODULAR MULTI-COMPONENT DEVS MODELS INTO MODULAR FORM

Multi-component systems with non-modular couplings, as discussed above, directly access and write on each other's state variables. In the sense of modular couplings, this means a violation of modularity. In the sequel, we give a procedure for translating non-modular coupled systems into modular form. Moreover, we will show that such a translation is always possible. The procedure works by identifying the dependencies between components and converting them into input and output interfaces and modular couplings. *Modularization* is the term used for such introduction of input/output interfaces.

We distinguish two forms of non-modular couplings which we have to treat differently. In case (1) component A has write access to a variable v of another component B (Fig. 7.3). In modular form, this has to be transformed into a communication from A to B through a modular coupling. The situation is modularized by giving A an output port, $vout$, and B an input port, vin, and by a coupling from $vout$ to vin. Whenever, in the original transition function, A wants to write to v, in the modular form, A has to initiate an output event at vout with the same output value. Through the coupling, this results in an input event at vin of B. The external transition function of B has to react by writing the received value into its variable v.

In case (2), component B has read access to a variable u of component A (Fig. 7.3). This situation is more complicated because B always has to have access to the actual value of the variable. The situation is solved in that B holds a copy of the state information of u in its own memory. Now, whenever A changes the value of u, B has to be informed of the change. This is accomplished by a coupling from

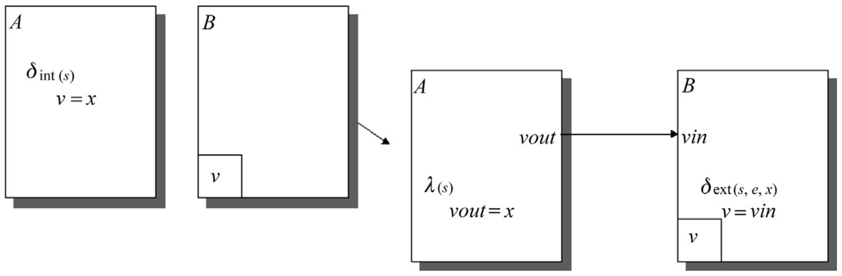


FIGURE 7.3

Component A has write access to variable v of component B.

A to B through ports $uout$ and uin in a similar way as in case (1). In this way B always has an actual value of variable u in its own state variable $ucopy$ (Fig. 7.4).

7.2.6 STATE UPDATING IN DISTRIBUTED SIMULATION

An important form of state updating occurs in distributed simulation, e.g., in the HLA run time infrastructure, as illustrated in Fig. 7.5. The components of a coupled model are mapped onto different computers, or nodes, in a network. To maintain knowledge of the states of other components, replicas of these components are retained on the node dedicated to a component. State updating, is employed to keep the local replicas, or proxies, up to date. In a more advanced concept, the proxies are not full copies of their counterparts but are smaller versions, that are valid simplifications (homomorphisms or endomorphisms) in the experimental frame of the simulation study.

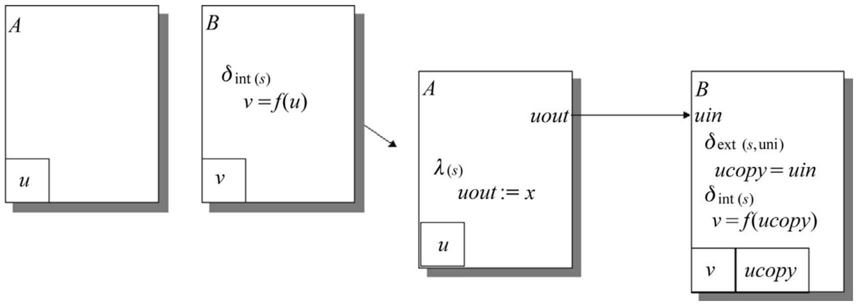


FIGURE 7.4

Component B has read access to variable u of component A.

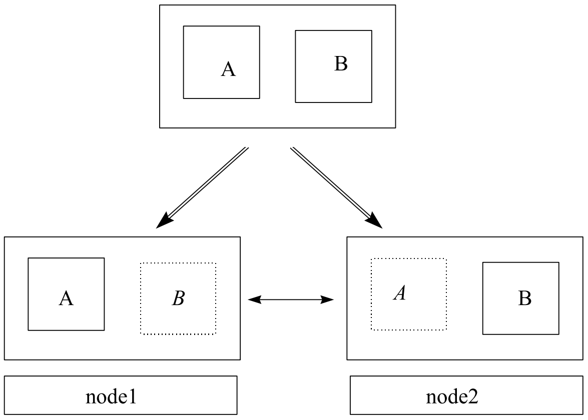


FIGURE 7.5

State Updating in Distributed Simulation.

7.3 DISCRETE TIME SPECIFIED NETWORK FORMALISM

In Chapter 3, we learned to know how to model digital sequential networks as built from elementary flip-flops and boolean gates. Digital networks are examples of networks of discrete time system components. In this section, we introduce the discrete time specified network formalism (DTSN) as a general discrete time formalism at the coupled system level. We specify discrete time networks as couplings of basic discrete time systems and of instantaneous functions, that is, memoryless models.

DELAY-FREE (ALGEBRAIC) CYCLES

Discrete time networks must obey certain constraints to fulfill closure under coupling. The most important constraint is that the network must be free of any *delay-less* (also called *algebraic*) loops, i.e., there is no cycle of output-to-input connections containing a component whose output feeds back to its input without going through some non-zero delay.

Fig. 7.6 will help to clarify. Suppose that we have a network model with three components A, B, and C, which are mutually interconnected. Additionally, A has an input coupling from the network input port. To compute the output of A we need the network input and the output from C. To compute the output of B we need the output from A. Finally to compute $C's$ output we need the output from B, thus forming a cycle of dependencies. Now suppose that there are no delays between input and output in any of the components. This means that all the components impose their input-to-output constraints on the ports at the same time. Then, given an input on the input port, we can try to solve for consistent values on the other ports. If there is an unique solution for every input then a consistent system can result. This approach is commonly adopted in differential algebraic equation models (Hairer and Wanner, 1991). However, we can obviate the problem by requiring that no algebraic loops are present. For example, we require that in the cycle of Fig. 7.6 there is at least one component, whose current output can be computed without knowledge of the current input.

Exercise 7.3. Give examples of choices for the components in Fig. 7.6 that create algebraic loops with consistent solutions. Do the same for inconsistent solutions.

Recall from Chapter 6, that in Mealy-type systems and in instantaneous functions, the output directly depends on the input as well as the state. Only Moore-type components compute their output

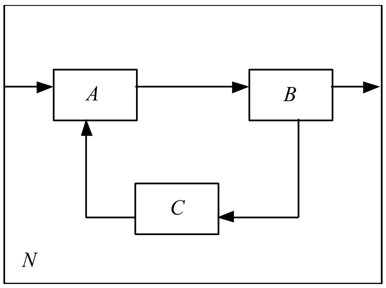


FIGURE 7.6

Dependencies in a network of DTSS.

solely on the state information and therefore impose a delay on the effect of the input on the output. *So to banish delay-less loops we require that in any feedback cycle of coupled components, there must be at least one component of type Moore.*

Exercise 7.4. Show that there are no delay-less loops if, and only if, the following is the case: when we remove all Moore type components then the remaining components form a set of directed acyclic subnetworks.

DEFINITION OF DISCRETE TIME COUPLED MODELS

A discrete time specified network (DTSN) is a coupled system

$$N = \langle X, Y, D, \{M_d\}, \{I_d\}, \{Z_d\}, h_N \rangle$$

where X, Y, D, $\{M_d\}$, $\{I_d\}$, $\{Z_d\}$ defined as in the general multi-component formalism in Section 5.7 and h_N is a constant time advance employed for the specification of the time base $hN \cdot \mathfrak{J}$. To be well defined, a discrete time network has to fulfill the following constraints:

- the components M_d are basic DTSS or FNSS,
- no delay-less cycles are allowed, i.e., in a feedback loop there has to be at least one component whose outputs can be computed without knowledge of its input or

  $\nexists d_1, d_2, ..., d_n \in D \cap \{d | d \in D \wedge d \text{ is a } FNSS \text{ or } d \text{ is of type } Mealy\} : d_1 \in I_{dn} \wedge d_i \in I_{di-1}$, $i = 2, ..., n$

- the time base h_N of the network and all its components have to be identical.

CLOSURE UNDER COUPLING OF DTSS

We now will show that the DTSS formalism is closed under coupling under the conditions just given. Recall, that for the detection of algebraic loops it is crucial to decide if a component is of type Moore or Mealy. Therefore, in the process of characterizing the resultant of a DTSN we will develop a procedure to tell whether it is of type Moore or Mealy.

Given a discrete time specified network DTSN, we associate with it the following basic discrete time specified system

$$DTSS_N = (X, Y, Q, \delta, \lambda, h)$$

where $Q = \times_{d \in D} Q_d, h = h_N,$ and $\delta : Q \times X \to Q$ and $\lambda : Q \to Y$ or $\lambda : Q \times X \to Y$ are defined as follows.

Let $q = (q_1, q_2, ..., q_d, ...) \in Q$ with $d \in D, x \in X$. Then $\delta((q_1, q_2, ..., q_d, ...), x) = q' = (q'_1, q'_2, ..., q'_d, ...)$ is defined indirectly through

$$q'_d = \delta_d(q_d, x_d)$$

with $x_d = Z_d(..., y_j, ...)$, where for any $j \in I_d$

$$y_j = \begin{cases} \lambda_j(q_j)) & \text{if } j \text{ is of type Moore} \\ \lambda_j(q_j, x_j)) & \text{if } j \text{ is of type Mealy} \\ \lambda_j(x_j) & \text{if } j \text{ is of type function, respectively.} \end{cases}$$

Note that y_j has a well-defined value in the second and third cases because of the assumption that no algebraic loops are present.

Now the output function, $\lambda : Q \to Y$ or $\lambda : Q \times X \to Y$, where λ is defined as $Z_N(..., y_j, ...)$, $j \in I_N$ and y_j defined as above.

At this stage, we do not yet have a way of deciding whether we have a function $\lambda : Q \to Y$ depending just on state q or it is a function $\lambda : Q \times X \to Y$ also depending on the input x, i.e., if the resulting system is of type Moore or type Mealy. However, having such a decision procedure is essential in hierarchical networks due to the requirement that no delayless cycles exist. Recall that this requires at that at least one Moore-type component to be present in all feedback loops. Thus, in hierarchical construction, we need to know whether a network, that we are encapsulating as a component, is of type Mealy or Moore. The following procedure enables us to decide. We examine the subnetwork of all Mealy and memory-less components. If in this network, we find an input from the network that is connected to an output to the network through Mealy-type components or instantaneous functions, then the network is of type Mealy, otherwise, it is of type Moore. (Since there are no algebraic cycles, the connection from the network to itself really represents a coupling from the external input to the external output.) Formally, we present this procedure in the theorem in the Appendix.

7.4 MULTI-COMPONENT DISCRETE TIME SYSTEM FORMALISM

Recall from Section 5.7 that multi-component systems employ a set I_d of influencing components and a set E_d of components itself influences. We have seen an example of a multi-component discrete time system – the Game of Life cellular automaton in Chapter 3. In this case, an individual component has a set of influencing components, called its *neighbors*. Its state transition function defines new state values only for its own state variables and does not set other components' states directly. It only influences them indirectly because the state transitions of the other components depend on its own state. Therefore, the next state function of the overall multi-component system can be visualized as having all the components look at the state values of their individual influencers and simultaneously computing the next state values for their state variables. We have already seen how this inherently parallel computation can be done on a sequential computer. In Chapter 8 and 10 we will formulate appropriate simulators for each of the formalisms that are amenable to both sequential and parallel computation.

A *Multi-component Discrete Time System Specification* is a structure

$$multiDTSS = \langle X_N, D, \{M_d\}, h_N \rangle$$

with

X_N is an arbitrary set of input values, h_N is the time interval to define the discrete time base, D is the index set.

For each $d \in D$, the component M d is specified as

$$M_d = (Q_d, Y_d, I_d, \delta_d, \lambda_d)$$

where Q_d is an arbitrary set of states of d, Y_d is an arbitrary set of outputs of d, $I_d \subseteq D$ is the set of influencers of d, $\delta_d : \times_{i \in I_d} Q_i \times X \to Q_d$ is the state transition function of d, and $\lambda_d : \times_{i \in I_d} Q_i \times X \to Y_d$ is the local output function of d. (The set of influencees E_d of d usually employed in multi-component models – see Section 5.7 – is defined implicitly to be the set $E_d = \{d\}$, having d as the unique element.)

In a multiDTSS, the set of components jointly generate the dynamics of the system. Each of the components owns its local state set, a local output set and local state transition and output function. Based on the state of the influencing components $i \in I_d$ and on the current input value, the component determines its next state and its contribution to the overall output of the system. In this way, the resultant of a multiDTSS is a DTSS that is built from the crossproduct of the components.

Formally, a $multiDTSS = \langle X, D, \{M_d\}, h_N \rangle$ specifies a $DTSS = \langle X, Y, Q, \delta, \lambda, h \rangle$ at the I/O system level as follows: $Q = \times_{d \in D} Q_d$, $Y = \times_{d \in D} Y_d$, $\delta(q, x)$ is defined by

$$\delta(q, x).d = \delta_d((...., q_i, ...), x),$$

$\lambda(q)$ is defined by

$$\lambda(q).d = \lambda_d((..., q_i, ...))$$

with $i \in I_d$, and

$$h = h_N$$

SPATIAL DTSS: CELLULAR AUTOMATA

Recall the cellular automata discussed in Chapter 3. Those can be formulated as special cases of the multi-component DTSS formalism. For example, consider a two-dimensional automaton such as the Game of Life. The multi-component DTSS model can be specified as follows:

$$multiDTSS = \left\langle X, D, \{M_{i,j}\}, h_N \right\rangle$$

X is the empty set since there is no input to the model, $h_N = 1$, since the time base is the set of integers, $\{0, 1, 2, ...\}$, $D = \{(i, j) | i \in I, j \in I\}$ is the index set.

For each $(i, j) \in D$, the component, $M_{i,j}$ is specified as:

$$M_{i,j} = \left\langle Q_{i,j}, Y_{i,j}, I_{i,j}, \delta_{i,j}, \lambda_{i,j} \right\rangle$$

where $Q_{i,j} = \{0, 1\}$, $Y_{i,j}$ is empty set since there are no outputs. The set of influencers $I_{i,j}$ is defined by its neighbors $(i, j+1)$, $(i+1, j)$, $(i, j-1)$, $(i-1, j)$, $(i+1, j+1)$, $(i+1, j-1)$, $(i-1, j+1)$, and $(i-1, j-1)$. Let

$$sum = q_{i,j+1} + q_{i+1,j} + q_{i,j-1} + q_{i-1,j} + q_{i+1,j+1} + q_{i+1,j-1} + q_{i-1,j+1} + q_{i-1,j-1}$$

be the sum of all neighbors of a cell at i, j. Then we define $\delta_{i,j} : \times_{k,l \in I_{i,j}} Q_{k,l} \rightarrow Q_{i,j}$ by

$$\delta_{i,j}(q_{i,j}, q_{i,j+1}, q_{i+1,j}, q_{i,j-1}, q_{i-1,j}, q_{i+1,j+1}, q_{i+1,j-1}, q_{i-1,j+1}, q_{i-1,j-1}) =$$
$$= \begin{cases} 1 & \text{if } q_{i,j} = 1 \wedge (sum = 2 | sum = 3) \\ 0 & \text{if } q_{i,j} = 1 \wedge (sum < 2 | sum > 3) \\ 1 & \text{if } q_{i,j} = 0 \wedge sum = 3 \\ 0 & \text{if } q_{i,j} = 0 \wedge sum \neq 3 \end{cases}$$

$\lambda_{i,j}$ is not specified since there is no overall output.

In general, any cellular automaton has a multi-dimensional geometrical grid structure which is formalized as an Abelian group. A *neighborhood* template is specified as a subset of the group and defines the influencers of the cell at the origin, or zero element, of the group. By uniformity, the influencers of the cell at any other point can be obtained by translating the neighborhood template to that point, i.e., $I_{i,j} = N + (i, j)$, where N is the neighborhood template. The cells all have isomorphic component structures: basically, the same state sets and the same state transition functions. In other words, a cellular automaton can be specified by one neighborhood template and one component structure.

Exercise 7.5. Show that the Game of Life multi-component formalization above can be put into the space invariant form just described. (Hint: relate the influencers of cell (i, j) to those of cell $(0, 0)$, hence to a neighborhood template and show that the transition function $\delta_{i,j}$, can be expressed as a function of state values assigned to the appropriate elements in this template.)

Exercise 7.6. Formulate a space-invariant DEVS cell space formalism analogous to the DTSS formalism for cellular automata just discussed. Express the discrete event form of the Game of Life in this formalism.

7.5 DIFFERENTIAL EQUATION SPECIFIED NETWORK FORMALISM

Networks of differential equation specified systems (DEVN) are analogous to discrete time networks and the same ideas apply. Let us introduce networks of differential equation specified systems in the following by translating the considerations of discrete time networks to the continuous domain.

Recall from above that discrete time networks are specified as modular couplings of basic discrete time systems and memoryless instantaneous functions. In the continuous domain we have basic differential equation specified systems and again instantaneous functions. The whole network describes a basic differential equation as the crossproduct of the continuous components where the input rates are defined based on the influencers' outputs.

We introduce *differential equation specified network* (DESN), analogous to discrete time networks, as a coupled system

$$N = \langle X, Y, D, \{M_d\}, \{I_d\}, \{Z_d\} \rangle$$

with the following constraints:

- X and Y have to be real vector spaces,
- the components $d \in D$ have to be DESS or FNSS, and

- no algebraic cycles are allowed, i.e., in a feedback loop there has to be at least one component the output of which can be computed without knowledge of its input.

CLOSURE UNDER COUPLING OF DESS

Given a differential equation specified network DESN, we are able to associate with it the basic differential equation specified system, $DESS_N$

$$DESSN = \langle X, Y, Q, f, \lambda \rangle$$

where $Q = \times_{d \in D} Q_d$, and $f : Q \times X \rightarrow Q$ and $\lambda : Q \rightarrow Y$ or $\lambda : Q \times X \rightarrow Y$ defined similar to δ and λ in DTSN as follows.

Let $q = (q_1, q_2, ..., q_d, ...) \in Q$ with $d \in D$, $x \in X$. Then $f((q_1, q_2, ..., q_d, ...), x) = q' = (q'_1, q'_2, ...q'_d, ...)$ is defined indirectly through $q'_d = f_d(q_d, x_d)$ with $x_d = Z_d(..., y_j, ...)$, $j \in I_d$ and $y_j = \lambda_j(q_j)$ or $y_j = \lambda_j(q_j, x_j)$ or $y_j = \lambda_j(x_j)$ if the component is of type Moore, of type Mealy, or an instantaneous function, respectively. The rest of the proof that f is well-defined follows along the lines of the closure proof for DTSS.

However, not only must f be well defined, it must also satisfy the Lipschitz condition.

$$||f(q, x) - f(q', x)|| \leq k||q - q'||$$

The proof that this is true will follow from the observation that f is composed of a sequence of coordinate functions, f_d, taking the form of the non-modular DESS specification to be discussed below. We will show that if each of the coordinate functions in the latter specification satisfies a Lipschitz condition, then so does the composite function. Since by assumption, in the present case, the f_d are the rate-of-change functions of well-defined DESS specifications, they do indeed satisfy the Lipschitz condition. Thus, on the basis of a proof yet to come, f satisfies the Lipschitz condition.

7.6 MULTI-COMPONENT DIFFERENTIAL EQUATIONS SPECIFIED SYSTEM FORMALISM

In a similar way to the DTSS case, we formulate a multi-component differential equation system specification, $multiDESS$, with non-modular coupling. Recall that the basic DESS formalism does not define a next state function directly but only through rate-of-change functions for the individual continuous state variables. In the multi-component case, the individual components define the rate of change of their own state variables based on the state values of their influencers. Let us first define the general formalism and then discuss the modeling approach by considering partial differential equations – a special model type showing much resemblance to the cellular automata in the discrete time domain.

A *multi-component differential equation system specification* is a structure

$$multiDESS = \langle X, D, \{M_d\} \rangle$$

where X is the set of inputs, a real valued vector space $\mathbb{R}^m$ and D is the index set. For each $d \in D$, the component M_d is specified as

$$M_d = \langle Q_d, Y_d, I_d, f_d, \lambda_d \rangle$$

where Q_d is the set of states of d, a real valued vector space $\mathbb{R}^n$, Y_d is the set of outputs of d, a real valued vector space $\mathbb{R}^p$, $I_d \subseteq D$ is the set of influencers of d, $f_d : \times_{i \in I_d} Q_i \times X \to Q_d$ is the rate of change function for state variables of d, $\lambda_d : \times_{i \in I_d} Q_e \times X \to Y_d$ is the local output function of d. The set of influencees E_d of d is again defined to be the set $\{d\}$. We require that each f_d satisfies a Lipschitz condition:

$$||f_d(q, x) - f_d(q', x)|| \le k_d ||q - q'||$$

In a multiDESS, the derivative function of each component defines the rate of change of its local state variables. Formally a $multiDESS = \langle X_N, D, \{M_d\} \rangle$ specifies a $DESS = \langle X, Y, Q, f, \lambda \rangle$ at the I/O system level in the following way: $Q = \times_{d \in D} Q_d$, $Y = \times_{d \in D} Y_d$, $f(q, x)$ is defined by

$$f(q, x).d = f_d((..., q_i, ...), x),$$

and $\lambda(q)$ is defined by

$$\lambda(q).d = \lambda_d((..., q_i, ...)),$$

with $i \in I_d$.

Now, we must show that the resultant derivative function satisfies the Lipschitz condition:

$$||f(q, x) - f(q', x)|| \le k ||q - q'||$$

This will follow from the fact that each of its coordinate functions satisfies such a condition by the constraint placed on these functions given before. We demonstrate how this works using two coordinates only:

$$\begin{aligned}
&||f(q_1, q_2, x) - f(q_1', q_2', x)|| \\
&= ||(f_1(q_1, q_2, x) - f_1(q_1', q_2', x), f_2(q_1, q_2, x) - f_2(q_1', q_2', x))|| \\
&\le ||f_1(q_1, q_2, x) - f_1(q_1', q_2', x)|| + ||f_2(q_1, q_2, x) - f_2(q_1', q_2', x)|| \\
&\le k_1 ||(q - q'|| + k_2 ||(q - q'|| \\
&\le (k_1 + k_2) ||q - q'||
\end{aligned}$$

7.6.1 SPATIAL DESS: PARTIAL DIFFERENTIAL EQUATION MODELS

Partial differential equation models emerge from an extension of differential equation where space coordinates, besides time, are introduced as independent variables. A partial differential equation specified system therefore shows variations in time as well as in space.

Partial differential equation systems require a science of their own and a whole discipline deals with the solution of such differential equation systems. We will only give an short glimpse of them here to place them into our framework of simulation modeling formalisms.

For our exposition, let us consider a simple example of a general flux-conservative equation in one variable u. The equation

$$\frac{\partial u}{\partial t} = -v\frac{\partial u}{\partial x}$$

expresses that the change of the variable u in time is equal to the negative velocity, $-v$, multiplied by the change of the variable u in the spatial dimension x. The result of this equation is a wave which propagates with velocity v along the x dimension.

The approach to solve such problems, which is a representative of the so-called *hyperbolic* partial differential equation, leads to the discretization of the space and time dimensions. Let us first introduce a discretization of space. The entire observation interval $[x_0, x_l]$ with length l is divided into k equal pieces each $\Delta x = l/k$ wide. Then we get k *mesh points* for which we set up equations to express the changes over time. In the so-called *Forward Time Centered Space* (FCTS) approach, this is done for each mesh point j by replacing the spatial derivative $\frac{\partial u_j}{\partial x}$ of u at point j by the difference of the neighboring states divided by the length of the spatial interval

$$\frac{u_{j-1} - u_{j+1}}{2\Delta x}$$

(note the similarity to the Euler integration method) giving an equation for the time derivative of variable u at point j

$$\frac{\partial u_j}{\partial t} = -v\frac{u_{j-1} - u_{j+1}}{2\Delta x}$$

for each mesh point j. Obviously, we have a multiDESS with k components and influencers set $I_j = \{j-1, j+1\}$ for each component j, as well as derivative functions as above.

Usually, when solving partial differential equations the model is set up by discretizing also the time dimension. When we apply the same difference method for discretizing the time dimension, namely dividing the time interval into intervals of equal length Δt, we can replace the time derivative of u at spatial point j and time point $n+1$ by the difference of the value at time $n+1$ minus the value at time n divided by Δ_t (Euler integration)

$$\frac{u_j^{n+1} - u_j^n}{\Delta t}$$

With that we finally obtain an equation for state at mesh point j for time $n+1$:

$$u_j^{n+1} = u_{j-1}^n - v\frac{u_j^{n-1} - u_j^{n+1}}{2\Delta x}\Delta t.$$

What have we accomplished finally? Starting with a partial differential equation with derivatives in time and space dimension, we have discretized space and time. With the discretization of space we obtained a continuous multi-component model in cellular form with equal derivative functions for the cells. With the discretization of space we finally have obtained a cellular automaton with neighborhood $\{j-1, j+1\}$, time step Δ_t, and equal next state function for cell j as above.

7.7 MULTI-COMPONENT PARALLEL DISCRETE EVENT SYSTEM FORMALISM

In the second edition parallel DEVS was introduced and applied to PDEVS networks, i.e., coupled models with modular components. As Vicino et al. (2015) point out, there was no discussion of how the equivalent parallel multi-component networks whose components are non-modular. Essentially, this means that we must omit the Select function from the specification and handle the resulting collisions when more than one component wants to change the state of another component. Recall that each component in a non-modular network can have two subsets of components: the influencers and influencees. When only one component is imminent it looks at the states of its influencers and uses them to compute new states for its influencees. So the question arises in the parallel case were there is no selection of one actor from the imminents: how should the desires of multiple imminents be resolved into unique next states for their collective influencees? Foures et al. (2018) define a multi-component Parallel DEVS formalism, called multiPDEVS, in which the imminents are considered to "propose" new states for their influences which have their own "reaction" function that implements an autonomous approach to uniquely resolving the multiple proposals. MultiPDEVS employs PDEVS constructs to collect state collisions in a bag and manage them explicitly without increasing message exchanges. Important points of the multiPDEVS formalism:

- A multiPDEVS network is shown to be equivalent to a well-defined atomic PDEVS model supporting hierarchical construction
- An abstract simulator is defined and provides implementation perspective.
- The CellSpace as studied by Wainer (Wainer and Giambiasi, 2001) can be considered as a restriction of multiPDEVS.
- implementation showed significant speedup for highly communicative models with tight coupling but also for modeling paradigms falling under bottom-up approaches.
- MultiPDEVS works at the model level as opposed to flattening of hierarchical structure (Bae et al., 2016) which produces direct coupling at the lowest level to reduce message routing. The non-modular approach eliminates I/O ports so that components are able to influence each other directly.

Extension of the multiPDEVS approach can produce a non-modular equivalent of the modular DTSS network (Section 7.5) which could support Multi-agent System modeling with multiPDEVS as agents and multiPDTSS as environment.

7.8 SUMMARY

After defining modular and non-modular coupled models and showing how a non-modular system can be transformed into modular form for DEVS, let us conclude this chapter with a comparison of the two approaches.

From the procedure for translating non-modular couplings into modular form, it is clear that non-modular systems are easier to build in the conception phase, because the whole system can be built as one gestalt without breaking up the interrelations of components. Components do not have to maintain information about their neighbors since they are allowed to access others' state information directly.

However, this deceptive advantage soon becomes undermined when one considers such issues as testing components or components' reusability. Components in modular coupled models are systems by themselves. From their interface description one has a clear understanding what the inputs to this systems can be and what one can expect as outputs. The input and output interfaces unambiguously show what each component needs to get from, and what it provides to other components. Components' dynamic behavior only depends on its inner state and its inputs. Such a module is much easier to comprehend and test for correctness. And modeling coupled systems in a bottom-up approach with well tested components is much less error-prone than developing complex systems as one big unit.

Also, modular system components lend themselves for reusability. Again the interface description gives a clear understanding where a component can be used. An interface-based classification of components (Thoma, 1990) is useful for organizing families of compatible components. We will come back to this question of reusability of simulation models when we discuss the organization of model bases and the architecture of modeling and simulation environments in the third part of the book.

Finally, modular coupled models best fit the needs of distributed simulation since their components can be assigned to network nodes in a straightforward manner.

The third edition added new Section 7.7 on multiPDEVS which fills the hole opened with the introduction of Parallel DEVS and its application in a modular but not non-modular way. The text summarizes (Foures et al., 2018) article which provides a comprehensive treatment of the introduced formalism addressing points such as well-definition and abstract simulators that are of interest whenever a new formalism is introduced.

Event routing and message sending overhead in DEVSRuby is reduced using techniques similar to Himmelspach and Uhrmacher (2006), and Vicino et al. (2015).

7.9 SOURCES

Kiviat (1971) was the first to characterize the different world views of discrete event simulation while Tocher (1969) was the first to provide a formal representation of the activity scanning world view. Nance (1971) characterized the time flow mechanisms for discrete event simulation. Renewed interest in activity scanning strategy is emerging in distributed simulation contexts (see Chapter 11). Process interaction world view discussions can be found in Cota and Sargent (1992) and Franta (1977). The hierarchical, modular approach to discrete event simulation was introduced by Zeigler (1984) and its relation to testing, reusability, and distributed simulation is discussed in Zeigler (1990).

APPENDIX 7.A

The following theorem gives the condition under which a network is of type Moore.

Theorem 7.1 (Characterization of Moore-type Networks). *Let $IMealy_d = \{e | e \in I_d$ and e is of type $Mealy$ or $Function$ or $e = N\}$ be those influencers in set I_d which are of type Mealy, Functions or the network itself and let*

$$IMealy_d^+ = IMealy_d \cup \cup_{e \in IMealy_d} IMealy_e^+$$

be the transitive closure of $IMealy_d$. Then the network N is of type Moore if and only if $N \notin IMealy_N^+$.

Proof. Let $IMealy_N^0 = \{N\}$ and for $i = 1, 2, ...$ let

$$IMealy_d^i = (\cup_{\{}e \in IMealy_N^{i-1}\}I_e) \cap \{d|d\ is\ of\ type\ Mealy\ or\ FNSS\ or\ d = N\}$$

be the indirect influencers of type Mealy of network output y N at level i. As there are no algebraic cycles allowed and the set of components is finite, $\exists i \in \mathfrak{J}^+$ such that $IMealy_N^i = \{\}$ and let n be the maximum i with $IMealy_N^i \neq \{\}$. Obviously,

$$IMealy_N^+ = \cup_{i=1,...,n} IMealy_N^i$$

For $i = 1, ..., n + 1$, let

$$IMoore_N^i = \cup_{e \in IMealy_N^{i-1}} I_e \cap \{d|disoftypeMoore\}$$

be the influencers of type Moore of the Mealy influencers at level $i - 1$. Then for $i = 1, ..., n$ and $\forall e \in IMealy_N^{i-1}$ it follows that $I_e \subseteq IMealy_N^i \cup IMoore_N^i$. Now, since $IMealy_N^{n+1} = \{\} \forall e \in IMealy_N^n$ we have $I_e \subseteq IMoore_N^{n+1}$.

/A) We first show if N is of type Moore, it is required that $N \notin IMealy_N^+$ and then from above follows $N \notin IMealy_N^i$ for $i = 1, ..., n$.

The output of network N is defined as a function Z_N dependent on the output of the Mealy and Moore influencers at level 1, $y_N = Z_N(..., \lambda_{d1}(q_{d1}, x_{d1}), ..., \lambda_{e_1}(q_{e1}), ...)$, for all $d_1, e_1 \in I_N$, $d1 \in IMealy_N^1$, $e_1 \in IMoore_N^1$. And for all $d_1 \in Imealy_N^1$ the output $\lambda_{d1}(q_{d1}, x_{d1})$ is computed using their input x_{d1}. However inputs x_{d1} are computed by functions Z_{d1} which are dependent on influencers of d_1 which are in the level 2 Mealy or Moore influencers of N, $x_{d1} = Z_{d1}(..., \lambda_{d2}(q_{d2}, x_{d2}), ..., \lambda_{e2}(q_{e2}), ...)$ for all $d_2, e_2 \in I_{d1}, d_2 \in IMealy_N^2$, $e_2 \in IMoore_N^2$. The inputs x_{d2} of the Mealy influencers d_2 are again computed in the same way using the influencers of d_2 which are in the level 3 Mealy and Moore influencers of N. We continue this process until we come to level n components' inputs. For these, however, the set of influencers of type Mealy is empty and the inputs are definitely only dependent on the Moore type components, $x_{dn} = Z_{dn}(..., \lambda_{en+1}(q_{en+1}), ...)$ for all $e_{n+1} \in I_{dn}$, $e_{n+1} \in IMoore_N^{n+1}$.

This shows that if $N \notin IMealy_N^+$, following the lines of computations of outputs of Mealy type influencers of the network output above, the network output y_N is finally only a function of the state vector $(..., q_d, ...)$, $d \in D$.

(b) The proof that, if $N \in IMealy_N^+$, the output of N is also dependent on the input x_N of N and the network therefore is of type Mealy, follows the lines of proof (a). □

REFERENCES

Bae, J.W., Bae, S.W., Moon, I.-C., Kim, T.G., 2016. Efficient flattening algorithm for hierarchical and dynamic structure discrete event models. ACM Transactions on Modeling and Computer Simulation (TOMACS) 26 (4), 25.

Chow, A.C.H., 1996. Parallel DEVS: a parallel, hierarchical, modular modeling formalism and its distributed simulator. Transactions of the Society for Computer Simulation 13 (2), 55–68.

Cota, B.A., Sargent, R.G., 1992. A modification of the process interaction world view. ACM Transactions on Modeling and Computer Simulation (TOMACS) 2 (2).

Foures, D., Franceschini, R., Bisgambiglia, P.-A., Zeigler, B.P., 2018. MultiPDEVS: a parallel multicomponent system specification formalism. Complexity 2018.

Franta, W., 1977. A Process View of Simulation. North-Holland, New York.

Hairer, E., Wanner, G., 1991. Solving Ordinary Differential Equations. II, Stiff and Differential-Algebraic Problems. Springer-Verlag, Berlin.

Himmelspach, J., Uhrmacher, A.M., 2006. Sequential processing of PDEVS models. In: Proceedings of the 3rd EMSS.

Kiviat, P.J., 1971. Simulation languages. In: Computer Simulation Experiments with Models of Economic Systems, pp. 397–489.

Nance, R.K., 1971. On time flow mechanisms for discrete system simulation. Management Science 18 (1), 59–73.

Thoma, J.U., 1990. Bondgraphs as networks for power and signal exchange. In: Simulation by Bondgraphs. Springer, pp. 9–40.

Tocher, K., 1969. Simulation languages. In: Aronsky, J. (Ed.), Progress in Operations. Wiley, NY.

Vicino, D., Niyonkuru, D., Wainer, G., Dalle, O., 2015. Sequential PDEVS architecture. In: Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium. Society for Computer Simulation International, pp. 165–172.

Wainer, G.A., Giambiasi, N., 2001. Application of the cell-DEVS paradigm for cell spaces modelling and simulation. Simulation 76 (1).

Zeigler, B.P., 1990. Object-Oriented Simulation with Hierarchical Modular Models. Academic Press, Boston.

Zeigler, B.P., 1984. Multifacetted Modelling and Discrete Event Simulation. Academic Press Professional, Inc, London.