On the formal specification of discrete event simulation models

Ulrich Freund
Volker Zerbe
Gunar Schorcht

March 27,1997

Abstract

In the early designphase of an eventdriven system one has to specify its functional behavior and its architecture. This can be done by modelling and simulation. In this paper, we present a formal description language to describe the function and the architecture by mapping extended finite state machines to expanded queueing systems, called HiFAQ's. The so specified models are translateted into a discrete event simulation program built up by a C++ class library.
Keywords: early design phase, extended finite state machine, expanded queueing system, discrete-event simulation

1 Introduction

In the early design phase of a discrete event system, one has to specify its behavior and its architecture. This can be performed by modelling the behavior as well as the architecture [1]. These models can be put to a discrete event simulation program.

Though there exists a lot of discrete event simulators based on queueing networks and finite state machines, there is no complete formal description available. Furthermore, one always has to live with some inconsitensies between the functional layer and the architectural layer. In particular, for mapping a functional description onto alternative architectural models one has to make changes in the whole model. Therefore, one needs a self consistent description for the functional part and architectural part of the model by keeping the identity of the events throughout the whole model.

This paper is organised as follows: After reviewing the basic descriptions for extended finite state machines (EFSM) and queuing systems (QS) we introduce expanded queueing systems (EQS) that hold the identity of events. We then introduce the mapping between an EFSM network and an EQS showing that these descriptions suffice to describe a system in a consistent manner.

2 EFSM's and QS

2.1 Requirements

A discrete event simulation system should be specified completly by a formal specification language hiding all implementation details. Typical constructs in a simulation language capable of modelling computer networks are queue manipulations, branching constructs, generation and deletion of events, sending and receiving different, valued types, and manipulation of the systems variables. Results are waiting time and throughput, the queuelength, number of corrupted packets and so on.

The constructs can be divided into two major classes:

1. Constructs to describe the behavior of the system
2. Constructs to describe the architecture of the system

To get the mentioned results in a computer network simulation, one needs both classes of constructs. In particular, the identity of messages in a queueing system has to be kept. Therefore, the model of queueing systems must be expanded to keep identity.

The description of the behavior with branching, sending and receiving of messages can be modelled with networks of extended finite state machines. A mapping between the expanded queueing models and the EFSM's has to be descriped. This consists of mapping the input/output messages to EFSM's as well as to provide information about the queueing system to the behavioural layer.

2.2 Existing Methodolgies

Here we describe the shortcomings of some popular modelling paradigms.

- **BONeS-Designer:**BONeS-Designer [5] uses for the behavioral description finite state machines and dynamic data flow graphs. Queues and Resources are available as C-Code Library. The lowest level is the dynamic dataflow graph which can consist of queues, FSM's or generic blocks. There is no distinction between behavior and architecture.
- **Opnet:**In Opnet[4] behavior can distinguished from architecture on the node level by means of special editors. However, one cannot built up a complete network constisting of merely finite state machines with a default architecture like SDL-simulators. Checking of the peer-to-peer behavior of a protocol stack is not easy. Parameter exchange between Node and Process level has to be done by C-Functions.
- **DEVS:** This is a formal description with no distinction between function and architecture. The way of describing the behavior is completly free, it can be done e.g. by Lisp or C-Code [6].
- **QPN:** Queueing Petri Networks [8] are a kind of formal description. Methods introduced here can be translated to QPN's. However, QPN's do not allow a distinction between functional and architectural modelling

In the rest of this paper we try to overcome the limitations of the above mentioned modelling methodogies.

2.3 Preliminaries

- $||$ is the concatenation operator
- $\mid S \mid$ denotes the number of elements in set S
- * is the Kleene operator on strings
- ϵ is the null string
- *Time(e)* is the event-time of event e
- $Proc_{Time}(T)$ is the processing duration for transition T
- P_f is a multiset [1]

2.4 EFSM's

An extended finite state machine (EFSM) extends an ordinary finite state machine by means of Variables. Variables can be of real or integer type. An EFSM is defined by a five-tuple [X,Z,Y,P,δ,λ] for the Mealy type and [X,Z,Y,Pδ,μ] for the Moore type. For the Mealy type we have:

- X is the input alphabet
- Z is the state set
- Y is the output alphabet
- P is the Variable set with p $\in \mathcal{P}$ or $\mathcal{R}$
- δ : X $\times$ Z $\times$ P $\longrightarrow$ Z $\times$ P is the state transistion where also parameters may be changed. function
- λ : X $\times$ Z $\times$ P $\longrightarrow$ Y is the output function

An extended finite state machine is used to describe communcitation protocols. One popular representive of this class is SDL.

[1] a multiset of a set M is a set where the elements of M may be multiply included. In contrast to a string, there is no ordering defined.

2.5 GSMS

A generalised semi markov scheme (GSMS) can represtent a timed-automaton (TA) descriped by a five-tuple [3] $TA = [E, H, \Gamma(H), f, G]$ for the case that we only consider deterministic state transitions.

- E is a countable event set
- H is a countable state set
- f is the state transition function with,

$$f(H, E) \rightarrow H$$

- and,
- $\Gamma(H)$ is the function of feasible events in state h $\in H$.
- G is a set of pdf's of interarrival times for each event.

In some sense, a FSM is a special case of a GSMS with the main difference that the state-space of a GSMS is countable infinite wheras the state-space of a FSM is finite. This difference is necessary e.g. to model queueing systems.

TA represents an automaton that is driven by a timed input stream

$$EL : \{e_1, e_2, ...\} = P_f(E)$$

Every element of the input stream has an associated time-stamp. Hence, considering only time, we have a time-sequecne

$$T_s : \{t_{1,e_1}, t_{2,e_2}, ...\} = \{Time_{e_1}, Time_{e_2}, ...\}$$

Such a system can only change its state at time-points t_i. The state-change is given by

$$h_{new} = f(h_{old}, \tilde{e})$$

with

$$\tilde{e} = min_{i \in \Gamma(h)}(t_i) \quad (1)$$
$$= min_{i \in \Gamma(h)}(Time_{e_i}) \quad (2)$$
$$= argmin_{i \in \Gamma(h)}(e_i) \quad (3)$$

During the state-transition new events may become feasible while others become infeasible. To describe this, we introduce the set of the new feasible events with $\pi(E) \subseteq P_f(E)$ and the set of unfeasible events $\tau(E) \subseteq P_f(E)$. The input-stream or eventlist (EL) of a TA changes during a transistion $f(h, \tilde{e})$:

$$\Gamma(h_{new}) : P_f(EL_{new}) = (P_f(EL_{old}) \setminus \tilde{e}) \setminus \tau(E) \cup \pi(E) \quad (4)$$

Equation 4 describes the behaviour of the eventlist.

2.6 From GSMS to Queueing-Systems

2.6.1 Queueing Systems

A queueing-system (QS) is a GSMS with $[E, H, f, \Gamma(h), G]$.

- E is the eventset with {a,d} where a denotes the arrival event and d denotes the departure event
- H is state-space with {1,2,...} and denotes the number of customers in the queue.

$$E = \{a, d\}$$

$$\Gamma(h) = \{a, d\} \forall_{h \geq 0}$$

$$\Gamma(0) = \{a\}$$

$$f(h, \tilde{e}) = \begin{cases} h+1 & \text{if } \tilde{e} = a \\ h-1 & \text{if } \tilde{e} = d \text{ and } h \geq 0 \end{cases}$$

$$G : pdfs$$

The state of the queueing system is the queue-length. The event-list consists of arrival and departure events. The time between two arrival events is called interarrival time and is denoted with λ. The time between the two consequetive departure events is called service time and denoted with μ. This might lead to some confusion with a Moore's FSA output functions. λ and μ are probabilistic values of arbitrary pdfs $g \in G$.

3 EQS

3.1 Expanded Queueing Systems

Queueing systems are an abstract representation of Queues with servers. For analysis, the identity of the events is lost. However, if we want to map FSM's or EFSM's to architectural models, we need to keep the identity of every incoming event. Therefore, we have to track the sequence of the events in the queue. This requirements leads to the Expanded Queueing Systems (EQS):

The state of an EQS is defined as follows:

$EQS = [E_e, H_e, f_e, \Gamma_e(H), G_e]$ The eventset E is defined by

$$E = E_a \cup E_d \tag{5}$$

The output signals are the released customers of the QS and therefore equivalent to the arrival eventset E_a. The state set H can be described as sequence:

$$H = [h_1, h_2, ...] = \{E_a\}^* \tag{6}$$

h_i defines the the place of the event in the queue.

Like the eventset, we can divide the statetransistion of the Queueing System by arrival transitions and by departarture transistion. In the case of an arrival event we have

$$f_a(H, E_a) = \begin{cases} H \| E_a & \text{if } \mid H \mid > 0 \\ h_1 & \text{if } \mid H \mid = 0 \end{cases} \tag{7}$$

In the case of a departure event we have to take the queueing strategy into account. In case of FIFO, the first element of the queue goes into service and, if the queue is not empty, all other elements go ahead one place in the queue.

$$f_d(H, E_d) = \begin{cases} H \setminus h_1 \wedge \forall_{i=2}^{n} h_{i-1} = h_i & \text{if } \mid H \mid > 1 \\ \epsilon & \text{if } \mid H \mid \leq 1 \end{cases} \tag{8}$$

The LIFO strategy looks like this:

$$f_d(H, E_d) = \begin{cases} H \setminus h_n & \text{if } \mid H \mid > 1 \\ \epsilon & \text{if } \mid H \mid \leq 1 \end{cases} \tag{9}$$

where n is the queuelength and h_n is the last element of the queue. h_i denotes the event on the i-th. position in the queue. The latter transistion functions also show that arrival events are queued whereas departure events release one element of the queue.

Hence, the complete state-transition function is:

$$f_e(H, E) = f_a(H, E_a) \cup f_d(H, E_d) \tag{10}$$

The outputfunction $\Gamma(H)$ depends only on the state of the EQS.

$$\Gamma_e(h) = \{E\} \forall_{h \geq 0} \Gamma_e(0) = \{E_a\} \tag{11}$$

The system time is the sum of the waiting time and the service time. The waiting time depends on the queueing-network architecture.

We can define the following mapping of an EQS to a QS. The arrival eventset of the EQS can be mapped to the generic arrival event

$$A : E_a \longrightarrow a \tag{12}$$

The same holds for the departure event.

$$D : E_d \longrightarrow d \tag{13}$$

Concerning the state, dropping the identity of the events in the queue, one can map the state of a queue to its lengths:

$$S : H_e \longrightarrow \mathcal{N} \tag{14}$$

$$F : E_e \longrightarrow E \tag{15}$$

The following section shows how a mapping between EFSM based functional description and the EQS based architectural description can be performed.

3.2 Mapping EFSM's on EQS'

The connection of a finite state machine with a generalised markov process can be done by a mapping. Doing this, we introduce the concept of time to a finite automaton. Every possible state input transition, i.e., every entry in the table of δ is assigned, a value. The value depends on experience and measurements. It is for example, the number of instructions a processor have to compute. Dividing the number of instructions by the intstructions per time, we will get the processing time [9]. The number of instructions per time is normally measured in MIPS, (Million Instructions Per Second).

From the queueing theory point of view, we have a queue with n different types of customers. N is the number of the elements in the input set. Let M be the number of states. Then we can assign every element of the input up to M-differnt time values. The actualy chosen depends on the current state. This is the service time μ for the i'th customer of type $e_a \in E_a = X_Z$ when the system is in state Z and signal X arrives.

$$\mu(X_Z) = Proc_{time}(\delta(Z,X,P)) \tag{16}$$

The mapping of the eventset E of the Queueing System and the input alphabet of the EFSM is done by dividing the eventset in arrival E_a and departure E_d events. Every element of the input alphabet of the EFSM is assigned a member of E_a. Additionaly, there are state transistions of the EFSM which cause new events, (e.g. setting a timer causing an interrupt later in time). Those additional events are also mapped to an element of E_a. To distinguish this event set from the input mapped event set E_a, we call this event set E_a. So we have

$$M : \delta(Z,X,P) \longrightarrow E_a \tag{17}$$

The set of departure events E_d are defined when a state-transition is served. For every service of an arrival event a departure event is generated after its associated transition. The time of the departure event relative to the system time is given by

$$Time_{e_d} = \mu(x_z) \tag{18}$$

When the departure event arrives the outputfunction of the EFSM fires:

$$P : \lambda(Z,X,P) \longrightarrow E_d \tag{19}$$

where x_z denotes the transition time for event X in state Z. Since for every event and state combintation a different processing time is possible, we have for the ordinary FSM

$$|E_d| = |E_a| = |X \times Z| \tag{20}$$

whereas for the EFSM, Equation 20 is given by

$$|E_d| = |E_a| = \Sigma_1^{NOS} NOT \tag{21}$$

NOT denotes the number of transistions per state and NOS the number of states. Transitions can be fired by the arrival of an input event or an state event. Every arrival event has its own service time, so that for every specific arrival event e_a an explicit departure event e_d exits. This means, we have as many different customers as E_a events. Every event therefore has an assigned service time.

The event time of a departure event e_d (see eq. 18) can be calculated iterative by equation

$$\mu(E_d) = Proc_{time}(\delta(Z,X,P)) \tag{22}$$
$$= Time(E_d) \tag{23}$$

The state of the Queueing system is defined by the queued arrival event and its place in the queue.

$$H = [h_1, h_2, ...] = \{\delta(Z,X,P)\}^* \tag{24}$$

h_i defines the type of the event and i is the place in the queue. This situation is depicted in Fig. 1

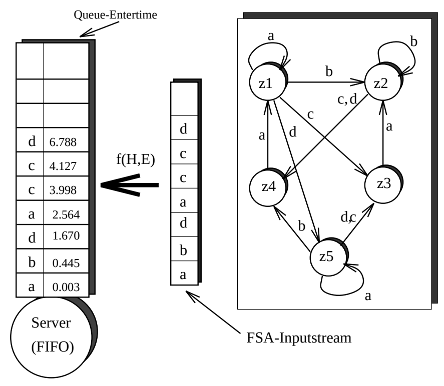

Figure 1: Mapping a functional description to an architecture

Like the eventset, we can divide the statetransistion of the Queueing System by arrival transitions and by departure transition. In the case of an arrival event we have

$$f_a(H, E_a) = \begin{cases} H \| \delta(Z, X, P) & \text{if} \mid H \mid > 0 \\ \\ h_1 & \text{if} \mid H \mid = 0 \end{cases} \tag{25}$$

The transition function for a departure event $e_d \in E_d$ is the same like Eq. 8 and Eq. 9 respectively. .

3.3 State-Param-Bind

In any system model, the state variables desribe the dynamic state of the systems. However, when there are interacting systems, the state transition of system A depends on the state of a linked system B [7]. During A's transition, the state of B acts like a parameter:

$$Z^A_{new} = \delta^A(X, Z_{old}, Z^B) \tag{26}$$

$$= \delta^A(X, Z_{old}, P) with P = Z^B \tag{27}$$

This property can be used to exchange information between an EQS and an EFSM. For example, the state of a queue, i.e., its length, can be passed to an EFSM which decides whether to queue or to drop an arriving packet. In this example, it suffices to use the QS form of the EQS.

4 Coupled Systems

From the horizontal point of view, a system can be divided into two levels, i.e., the functional and the architectural level. The functional level is a network of EFSM's whereas the architectual level is a network of EQS. This is called horizontal grouping. To minimise the coupling links and to allow variable structure systems, we propose vertical grouping. We call this vertical grouping HiFAQ (Hierarchical Functional Augmented Queueing networks). The coupling between HiFAQ's carry information from the functional level as well as from the architectural level.

A HiFAQ is defined by:
Hif = $[I, O, SM, PSC, \Psi]$ where

- I: Input ports
- O: Output ports
- SM: Submodels
- PSC: Parameter State Couplings
- Ψ: Coupling function between the SM's

A Submodel is defined by:
SM = $[I, O, M, \kappa]$ where

- I: Input ports
- O: Output ports
- M: Models $\{EFSM, EQN\}$
- κ: Mapping of the inputsignals to the input ports, the outputsignals to the output ports.

4.1 Inter HiFAQ communication

A discrete event model can be divided into a network of HiFAQ's. A Model is defined as:

$$Model = [Hif, K_B]$$

K_B is the coupling between the HiFAQ's. K_B is a three dimensional Matrix. A twodimensional matrix describes which type of signals are sent between the HiFAQ's whereas the third dimension describes which HiFAQ's are coupled.

$$K_B[k, l, x] = \begin{cases} 1 & \text{if signal x of HiFAQ k} \\ & \text{ is sent to HiFAQ l} \\ 0 & \text{otherwise} \end{cases} \tag{28}$$

The functional level within a HiFAQ can consist of more than one EFSM. The same holds for the architectural level consisting of more than one EQS.

4.2 Intra HiFAQ communication

Almost the same scheme is used for the Intra HiFAQ communication. Each submodel, i.e, an EFSM or an EQS, has its own two dimensional matrix with the same format like the Ψ matrix of the HiFAQ's. However, to send a signal from one HiFAQ to another one, one has to couple this link from a submodel's output port to the HiFAQ's output port. The coupling has to be done on the input side of the receiving HiFAQ. Therefore, every HiFAQ has an additional twodiminsional input mapping matrix. The output matrix of a HiFAQ represents a combination of the Ψ and κ matrix. The value of the κ matrix is the index to the Ψ matrix.

$$\Psi_j[o,x] = \Psi_k[i,\kappa[i,x]] \tag{29}$$

where $j,k \in Hif, i \in I, o \in O, x \in X$ in the case of an EFSM. The κ value hold the submodel SM of HiFAQ k.

5 A Compiler

To describe a discrete event simulation model by means of HiFAQ's we introduce a HiFAQ description language. Using this language, one first has to describe the signals, than the HiFAQ's and at last the couplings between the HiFAQ's. The internal couplings of the HiFAQ's are described within the HiFAQ.

A typical HiFAQ has the following form:

```
hifaq <name>
input <inputname1>
input <inputname2>
parambind : <SM.P> -> <SM.S>
models:
fsa <fsaname>
input <inputname1> : <signal1>,<signal2>
state : <state1>,<state2>,<state3>
output <inputname1> : <signal1>,<signal2>
delta:
  <state1> -> <state2> : <signal1>
  <state2> -> <state3> : <signal2>
  <state3> -> <state1> : <signal1>
lambda:
  <state1> > <signal1> : <signal2>
  <state2> > <signal2> : <signal1>
end fsa

qs <qsname>
input <inputname1> : <signal1>,<signal2>
state : <state1>,<state2>,<state3>
output <inputname1> : <signal1>,<signal2>
f:
 fifo
end qs
end hifaq
```

A signal has the form of a record definition in pascal. A typical signal for a frame looks like this

```
fsasig extern frame
{
  length:real;
  source: hifa;
}
```

The distinction between internal and external signals shall clarify the relationsships of the system.

6 Summary

In this paper we showed how well know system theoretic modelling approaches can be used to describe a discrete event simulation model. A HiFAQ description language has been formulated and a compiler for translating a so described model into a C++ program has been developped. However, a simulation framework has to be developed before a broader communinty can gain access to this simulator.

References

[1] R. Gerlich, *Early Validation of System Design by Behavioural Simulation*, ESTEC 3rd Workshop on 'Simulation for European Space Programs', Noordwijk, 1994

[2] D. Hogrefe *SDL, Estelle und Lotos* Springer-Verlag Heidelberg 1988

[3] C. Cassandras *Discrete Event Systems*, Asken, Homewoold1993

[4] Mil3 *Opnet User Manual* Washingtion DC, 1992

[5] Cadence Alta Group *BONeS-Designer Modelling Handbook* Sunnyvale 1995

[6] B. Zeigler, *Object Oriented Simulation with Hierarchical Modular Models, Academic Press 1990*

[7] M. Krapp*Digitale Automaten* Verlag Terchnik, Berlin 1988

[8] F. Bause *Queueing Petri Nets*, Fifth Int. Workshop on Petri Nets and Performance Models, Toulouse, October 1993

[9] C. Passerone et al. *Evaluation of trade-offs in the design of embedded systems via co-simulation* Technical Report, University of California at Berkely, Berkeley 1996