

Chapter 5

Specification and Description Language for Discrete Simulation

Pau Fonseca i Casas

Universitat Politècnica de Catalunya - Barcelona Tech, Spain

ABSTRACT

Designing a new simulation model usually involves the participation of personnel with different knowledge of the system and with diverse formations. These personnel often use different languages, making more difficult the task to define the existing relations between the key model elements. These relations represent the hypotheses that constrain the model and the global behavior of the system, and this information must be obtained from the system experts. A formalism can be a powerful tool to understand the model complexity and helps in the communication between the different actors that participate in the definition of the model. In this chapter we review the use of the “Specification and Description Language,” a standard and graphical language that simplifies the model understanding thanks to its modular nature. To do this we present a complete example, representing a simple queuing model that helps the reader to understand the structure and the nature of the language.

INTRODUCTION

The principal motivation of Operations Research (OR) is to understand the behavior of systems through representative models. Alternatives can be evaluated without interacting with or disturbing reality to choose the most appropriate system modification. Models can be used not only to compare alternatives but also to predict the

DOI: 10.4018/978-1-4666-4369-7.ch005

behavior of a known system when variables are modified within a specific range. Hence, operations research simulation models can yield solutions for possible future situations. The simulation field contains a variety of different paradigms to model a real system; this study uses a specification and description language to represent discrete-event simulation models. Based on (Law & Kelton, 2000) Law & Kelton (2000), Guasch, Piera, Casanovas, & Figueras (2002)(Guasch, Piera, Casanovas, & Figueras, 2002), and (Fishman, 2001) Fishman (2001), discrete-event simulation consists of three major methodologies: process interaction, activity scanning, and event scheduling.

The event-scheduling methodology is based on an initial description of events. Events are the elements of the model that cause modifications in the state variables of the model. A function is defined for each event. The functions are executed when the time associated with an event is the same as or very close to that shown on the model clock¹. The events are sorted in an event list by their time and priority. The time between two events is irregular. The simulation clock therefore jumps from one time to the next without following any set pattern (Law & Kelton, 2000).

To model a system that is composed of a server that receives elements over time according to a specific distribution, different events can be generated that represent the elements entering the system if the intervals between the events are known. Similarly, if the distribution that defines the time that the server requires to process an element is known, an event that defines the service time for a specific element can be generated. This information is illustrated in Table 1.

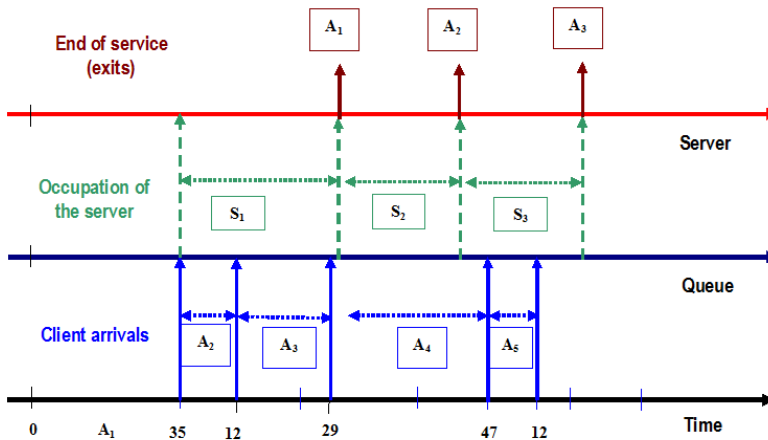
From Table 1, the diagram (time chart) shown in Figure 1 can be generated.

This figure shows an M|M|1, following Kendall's notation (Kendall, 1953), is a representation of a system composed by one machine proceeded by a queue (FIFO)². The initial state of the server is free, and the events shown in Table 1 define the behavior of the model. In the event-scheduling paradigm, one event is processed in each simulation loop and the procedure related to this event is executed. The pro-

Table 1. Events

Time between Arrivals		Service Time	
a1	35	b1	40
a2	12	b2	30
a3	29	b3	30
a4	47	b4	20
a5	12	b5	30

Figure 1. Event-scheduling time chart



cedure definition and the event-selection procedure determine the behavior of the model. The specification and description language (SDL) is based on the description of the process that defines the action after the different simulation elements receive a specific event (SIGNAL in the realm of SDL). Therefore, this model is a discrete simulation paradigm, which is similar to an event scheduling approach.

DESCRIPTION OF SDL

History of SDL

The motivation behind presenting the history of SDL is to understand its evolution and to understand that it is an evolving language that attempts to meet the needs of the professionals that work with it. A good review on the history of SDL can be found in (Reed, SDL-2000 new presentation, 2000)Reed (2000), which is an online PowerPoint presentation that includes comments and is an excellent review of SDL-2000.

SDL is an object-oriented, formal language that is standardized by the International Telecommunication Union, Telecommunication Standardization Sector (ITU-T) (formerly Comité Consultatif International Télégraphique et Téléphonique [CCITT]) as Recommendation Z.100 (ITU-T, 1999). The language is designed to specify complex, event-driven, real-time, interactive applications that involve many concurrent activities using discrete signals to enable communication (Doldi, 2003;

ITU-T, 1999) (Doldi L., 2003). Additionally, SDL can be easily used in combination with UML.

The development of SDL began with an ITU study in 1968. The main concern of that study was to find a way to handle stored program control switching systems. The primary finding in that study (in 1972) was that it is necessary to develop languages for the specification of complex systems. In 1976, the first SDL standard, the Orange Book, which contains a simple graphical language, was produced.

In 1980, the Yellow Book, which contained the semantics for the processes, was published. Then, in 1984, the Red Book appeared, defining the structure and the data. The Red Book also included a more rigorous definition of the tools that were necessary to support the language. Some additional tools were developed that same year. The development of the first SDL tools significantly changed the evolution of SDL because the tools forced both the users and the designers to be more formal. Increasing the level of formalism implies an increased level of work that is required to define the language. However, some benefits arise from the formalism, such as the identification of errors in the syntax of the model and the capability to vary parameters to perform effective model simulations. In the early 1980s, computer graphics were just starting to be commonplace, and the prices of computers were becoming reasonable. Therefore, the tools to incorporate a graphical facility to simplify interactions with the user were developing.

Another interesting aspect is that because the users want to evaluate “what-if” scenarios, it is necessary to develop a program from the SDL representation of the model that can provide the different alternatives. In that sense, it is common that some SDL tools, such as Cinderella (CINDERELLA SOFTWARE, 2007) or PragmaDev (PragmaDev SARL, 2012), among others, allow for the automatic generation of code, usually C or C++ code, from the SDL representation of the model³.

In 1988, the Blue Book was published (SDL-88), with well-defined syntax and a formal definition of the language. Additionally, effective tools were developed.

The next version, SDL-92, included object-oriented types; in other words, blocks and processes could be assigned types, with associated inheritance and parameterization of the object instances. In 1996, an addendum was published, which added minor modifications to the language.

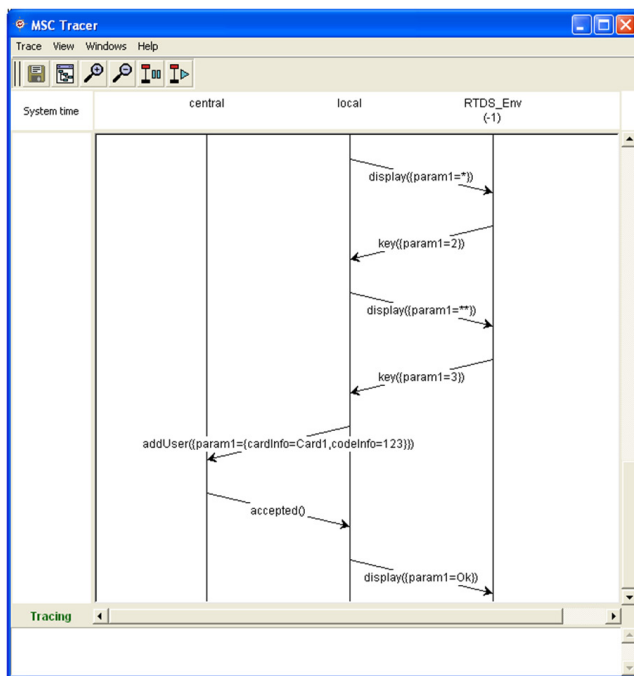
With the creation of tools that understand SDL, increased formalism is required to make the models capable of solving “what-if” questions. Solving “what-if” problems is an example of how validation and verification processes come together to achieve a common objective, that the tool and the model represent exactly what we want them to represent.

The message sequence chart (MSC) is an interaction diagram from the SDL family (see Figure 2). In concept, the MSC is similar to UML’s sequence diagram, and

it is standardized by the International Telecommunication Union. The definition of SDL lacks information describing how the signals travel through the system; MSC defines the communication behavior in real-time systems; specifically, telecommunication electronic switching systems.

In 1992, MSC appeared as a separate standard. MSC had been proposed as a standard in 1989, and then, in 1996, major additions to the standard were published. Some studies have explained how to harmonize UML sequence diagrams and MSC (Rudolph, Grabowski, & Graubmann, 1999). The 1996 version introduced high-level message sequence charts (HMSC) (ITU-TS, 1997), the MSC system for expressing state diagrams. In the latest version of MSC (2000) (ITU-TS, 2004), object orientation was included, the use of data and time in the diagrams was refined, and the concept of a remote method call was added. SDL-2000 was designed to address two important issues. One requirement was the ability to link with object modeling, specifically UML, and the other was to improve the implementation ability of SDL or, more specifically, to simplify the automatic generation of an implementation through code generation. The changes implemented to meet those requirements

Figure 2. MSC obtained from PragmaDev MSC Tracer © [2013 [PragmaDev]. Used with permission.



include exception handling and the introduction of textual algorithms to be used on diagrams. More details on SDL-2000 can be found in (Reed, SDL-2000 form New Millenium Systems, 2000) Reed (2000).

The upcoming version of SDL (SDL-2010) improves the language in many ways; specifically, for the scope of discrete simulation, it allows the definition of delays and priorities implicit in the signals, thus simplifying the representation of the dynamic behavior of the model.

One of the main attributes of SDL is that it can be considered to be a bridge between specification and implementation because it supports modeling at an abstract level but also details a complete unambiguous description of the implementation. One of the strengths of SDL is that the users have the power of a clear and formal textual syntax, and another strength is the ease-of-use that is provided by having a clear graphical representation.

SDL is also related to other ITU-T standards and, with MSC, provides a complete solution for the specification and design of systems. SDL can also be integrated with ASN.1 for protocol definition and with TTCN for validation and testing.

ITU Standards

The International Telecommunication Union (ITU, from French: Union Internationale des Télécommunications) is a United Nations special agency. This agency is responsible for information and communication technologies, coordination of the shared global use of the radio spectrum and satellite orbits, development of technical standards, and improvements to the access to information technologies by world communities. ITU has always been an intergovernmental public/private partnership organization. Membership currently includes 191 countries (member states)

Figure 3. Main building of the UIT-ITU in Geneva (Switzerland)



and more than 700 public and private sector companies. International and regional telecommunication entities (sector members and associates) are also represented.

The standardization work of ITU started with the birth of the International Telegraph Union, which became a United Nations special agency in 1947. The International Telegraph and Telephone Consultative Committee (CCITT, from French: Comité Consultatif International Téléphonique et Télégraphique) was created in 1956. This agency was renamed ITU-T in 1993.

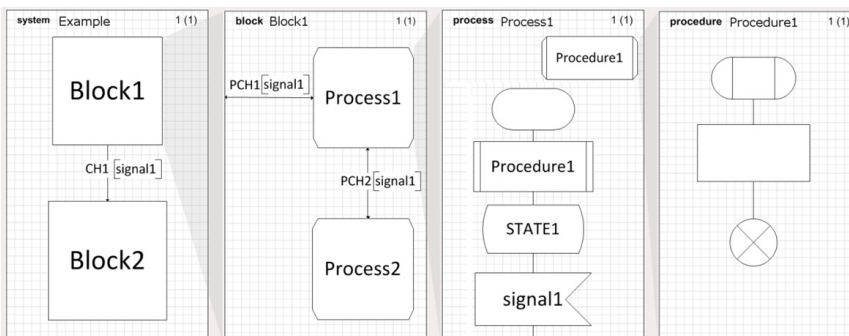
The ITU Telecommunication Standardization Sector (ITU-T) is one of the three divisions of the International Telecommunication Union (ITU). The main mission of ITU-T is the coordination of standards for telecommunications.

The Telecommunication Standardization Bureau (TSB) is the permanent secretariat of the ITU-T. It is located at the ITU headquarters in Geneva, Switzerland (see Figure 3).

ITU is one of the most universally well-recognized information and telecommunications standards disseminators. As an example, in 2007, ITU-T produced over 160 new and revised standards (named ITU-T Recommendations).

All of the ITU standards are called “recommendations” because they recommend standards to national bodies. The generation of these standards is the result of collaborative work. These standards are unambiguous, clear, precise, easy to use, easy to communicate, and easy to learn. The standards also support analysis, modeling, abstraction, and, importantly, product development, such that several tools currently understand the language. SDL and MSC have become international standards that are well-supported by many commercially available tools (as of 2000: Cinderella from Cinderella [CINDERELLA SOFTWARE, 2007], SDL, IBM SDL Suite [IBM, 2009], and PragmaDev RTDS[PragmaDev SARL, 2012]). A UML profile based on SDL has been standardized by ITU-T: Z.109.

Figure 4. SDL levels



Specification and Description Language

To define a simulation model, its structure and its behavior must be defined. SDL allows the user to define the structure, behavior, and other aspects, as follows:

- **Structure:** System, blocks, processes and hierarchy of the processes.
- **Behavior:** Defined through the different processes.
- **Data:** Based on abstract data types (ADT).
- **Communication:** Signals, and the parameters and channels that the signals use to travel.
- **Inheritances:** Used to describe the relationships between, and specialization of, the model elements.

Figure 5. System diagram (Doldi L., 2003). Three BLOCKS compose this model, block1, block2 and block3. The communication mechanism between them is represented by the channels. In this case, delaying channels are used, which are called DLCaSU, DLCbSU and DLCaDL.

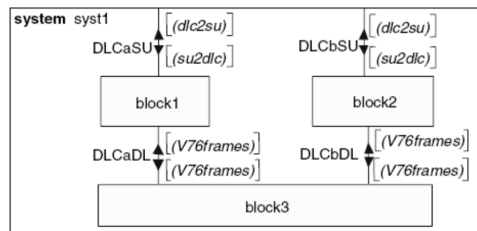
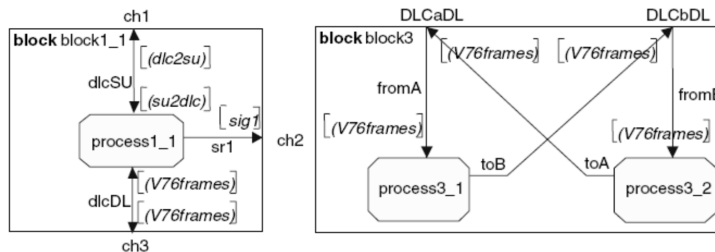


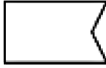
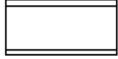






Figure 6. SDL block diagram (Doldi, 2003)



Specification and Description Language for Discrete Simulation

Table 2. Some important SDL blocks used in *PROCESS* diagrams

	Start. Defines the first operation to be executed at the initial state of a process. This operation can be used to define the initial conditions of a simulation model.
	State. A state element contains the name of a state. This element defines the states of behavioral diagrams (such as <i>PROCESS</i> diagrams).
	Input. Input elements describe the types of events that can be received by a process. All of the branches of a specific state start with an Input element because an object changes its state only when a new event is received.
	Create. This element allows the creation of an <i>SDL AGENT.c</i>
	Task. This element allows the interpretation of informal text, semi-formal actions (C code), or formal actions (SDL action language). In this chapter, following <i>SDL-RT</i> (PragmaDev SARL, 2006), we use C code here.
	Procedure call. These elements perform a procedure call. A <i>PROCEDURE</i> can be defined in the last level of the <i>SDL</i> language. It can be used to encapsulate pieces of the model for its reuse.
	Output. These elements describe the type of <i>SIGNAL</i> to be sent, the parameters that the signal carries and the destination or the <i>CHANNEL</i> the <i>SIGNAL</i> must follow. Other attributes of the event can also be detailed (e.g., priority, delay).
	Decision. These elements describe bifurcations. Their behavior depends on how a related question is answered.

The language is modular, allowing either top-down or bottom-up modeling and simplifying the validation of complex systems. The language has 4 levels: (i) *SYSTEM*, (ii) *BLOCKS*, (iii) *PROCESSES* and (iv) *PROCEDURES*, as shown in Figure 4. *SYSTEMS*, *BLOCKS* and *PROCESSES* are called *AGENTS*.

SDL System Diagrams

System diagrams represent all of the objects that make up a model and the communication channels between them. A *SYSTEM* is the outermost agent that communicates with the environment. Figure 5 shows a system that contains three blocks [12].

SDL Block Diagrams

The next stage in *SDL* specification is the construction of a block diagram for each of the different *BLOCKS* defined in the system diagram. Figure 6 represents the

block diagram for the block1 and block3 elements defined above (Doldi, 2003(Doldi L., 2003)).

Each rectangle represents a BLOCK. The lines that join the BLOCKS are the communication channels (bidirectional or unidirectional communication elements). The channels are joined to the objects through ports. Ports are important elements for implementing and reusing objects because they ensure the independence of the different objects. An AGENT only knows its own ports, which are the doors through which it communicates with its environment.

Each BLOCK has a name specified by the block keyword. The block diagram contains a number of PROCESSES and can also contain other BLOCKS (also mixed with PROCESSES in SDL-2000). PROCESSES communicate via channels (in versions previous to SDL-2000, signal routes can be used), which connect to other PROCESSES or channels external to the BLOCK.

SDL Processes

The PROCESSES is an AGENT that describes the behavior. Each of the PROCESSES of an AGENT has one or more states. For each of the states of a PROCESS, SDL describes how the state behaves depending on the different SIGNALS that can be received (events from the point of view of a discrete simulation). An AGENT can

Figure 7. SDL process diagram (Doldi, 2003)

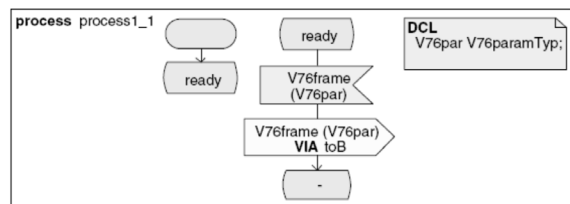
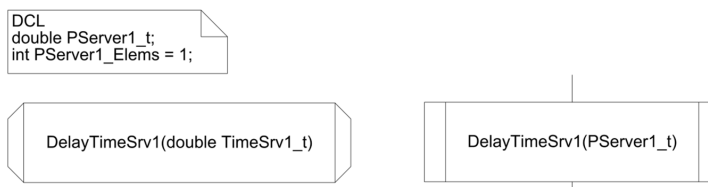


Figure 8. SDL PROCEDURE and PROCEDURECALL. C is used to represent the code of the SDL model following SDL-RT (PragmaDev SARL, 2006).



also react differently to a SIGNAL, depending on the port that sends the SIGNAL. The PROCESS is defined using graphical elements that describe operations or decisions. Table 2 shows some of the elements used in the SDL PROCESS diagrams to represent the model behavior. Figure 7 shows an example of an SDL PROCESS.

SDL Procedures

The last level of the specification and description language is the definition of the different procedures that appear in the SDL diagrams.

PROCEDURES allow parts of the PROCESS to be encapsulated to increase the readability. As an example, Figure 8 shows a PROCEDURE called DelayTimeSrv1 that calculates the time needed to perform an operation. The procedure is declared in the PROCEDURE element and can be used throughout the PROCESS with the PROCEDURECALL block.

Often, from the point of view of model understanding, it is not necessary to define what occurs inside PROCEDURES. However, to obtain the code or to perform

Figure 9. Relation between the non-graphical SDL and the graphical SDL

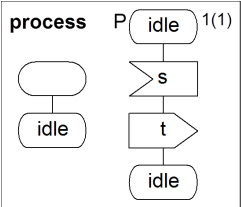


Figure 10. Non-delaying channels

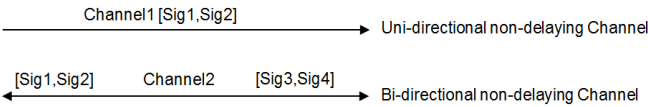
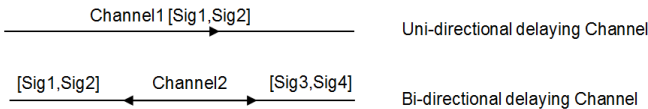


Figure 11. Delaying channels



an execution from the SDL specification, PROCEDURES must be completely defined.

The diagrams assist in the description and the specification of the model, detailing the most important aspects at the necessary level, depending on the target requirements.

To learn more about SDL, recommendation Z.100 (ITU-T, 1999) can be consulted, or information can be reviewed online at www.sdl-forum.org or in (SDL Tutorial) SDS Tutorial (Reed, Re: SDL-News: Request for Help: Initialisation of Pids, 2000) or (Doldi L., 2003)Doldi (2003), among other sources.

SDLP/PR

A textual SDL exists (see an example in Figure 9). This non-graphical SDL is not used in this study. The representation capabilities of the two SDL forms are equivalent (ITU-T, 1999).

SDL Channels

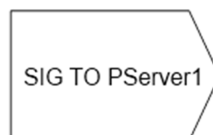
An SDL CHANNEL is the path that the SIGNALS follow to reach their final destination. They connect AGENTS between them or AGENTS with the environment. They can be uni- or bi-directional, they can have identifiers, and they can retain lists of all of the signals that they carry.

From the point of view of the management of the SIGNALS that a CHANNEL receives, a CHANNEL is a path that, depending on its nature, might introduce a delay.

Figure 12. SIGNAL declaration



Figure 13. Explicit signal routing using TO



Specification and Description Language for Discrete Simulation

Two main types of channels exist, which are the delaying and non-delaying channels. Non-delaying channels do not introduce any delay in the transmission of the SIGNALS. Thus, the SIGNALS reach their destinations immediately (at least from the point of view of the model's logic).

The representation of non-delaying channels is shown in Figure 10.

The delaying channels, as shown in Figure 11, introduce a delay in the transmission of the SIGNAL. The delay cannot be defined in SDL because, from the point of view of a discrete simulation, delaying channels cannot be used. It is necessary to completely control the model representation of the delays, as shown in Table 1.

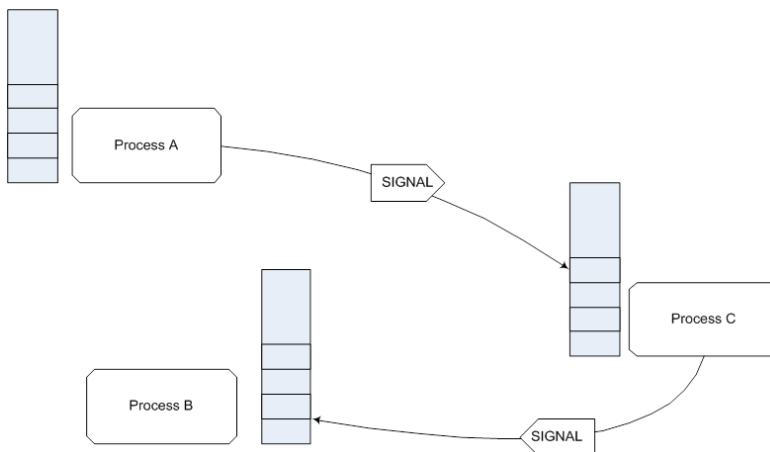
In the following section, the representation of time in SDL discrete simulation models will be described.

SDL Signals

SIGNALS in SDL represent the events that trigger the behavior of the simulation model. Each of the SIGNALS must be defined in a text area, as shown in Figure 12.

The declaration of a signal includes the type of signal and identifies whether the signal carries a parameter. Once the SIGNALS are defined, they are assigned to different CHANNELS in the model. It is easy to define the paths that the events must follow in the diagrams; however, sometimes a SIGNAL that must be sent from one PROCESS to another PROCESS can use different channels to reach its destination. Additionally, it is possible that different CHANNELS share the SIGNALS that

Figure 14. All of the SDL PROCESSES have a FIFO queue to process the SIGNALS that arrive at the AGENT.



can travel through them; hence, an addressing mechanism is needed to avoid sending a SIGNAL to a different PROCESS than the target.

To accomplish the foregoing steps, SDL allows the user to define the destination of an OUTPUT in several ways. If nothing is specified, then the destination is implicit. Usually, the use of an implicit destination occurs when only one destination is possible. If more than one destination is possible, then one of the PROCESSES receives the SIGNAL; however, it is not certain which of the possible destinations actually receives the SIGNAL. To avoid this ambiguity, it is preferable to use explicit addressing.

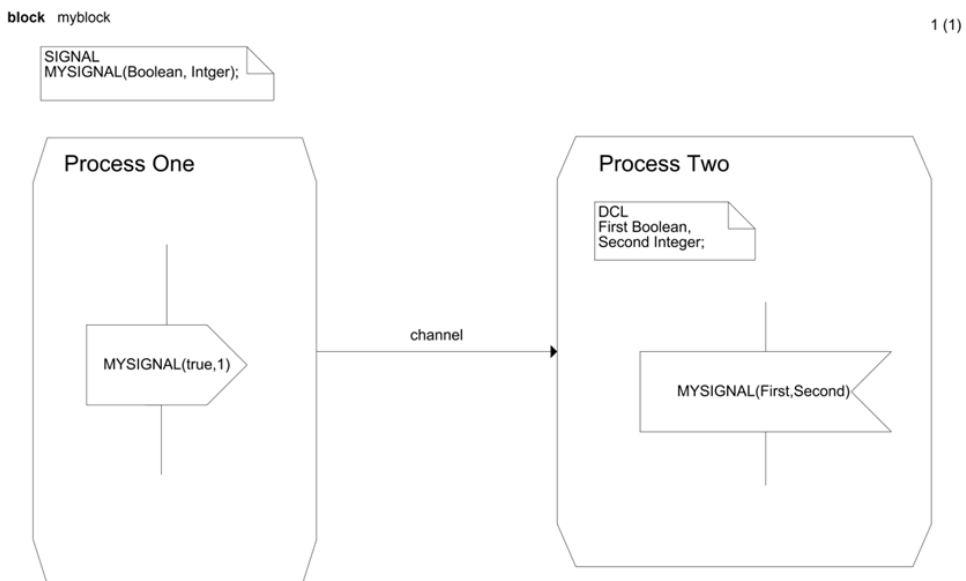
With explicit addressing, we can specify a PId (process identifier) that unequivocally identifies a PROCESS in the model (i.e., the PROCESS that must receive the SIGNAL). In other words, the keyword to can be used to identify the PROCESS that must receive the SIGNAL (see Figure 13).

With respect to the PId, no rules exist to define this identifier; SDL requires only that all of the PROCESSES in the model have a unique PId.

Four keywords obtain specific PIds from the model, as follows:

- **SELF:** Returns the PIs of the PROCESS itself.
- **SENDER:** Returns the PIs of the PROCESS from which the last SIGNAL came.

Figure 15. Passing parameters in the SIGNALS. In the figure, two PROCESSES are sharing information through the parameters of the SIGNALS.



Specification and Description Language for Discrete Simulation

- **OFFSPRING:** Returns the PIs of the last PROCESS that was created by the CREATE element in the current PROCESS.
- **PARENT:** Returns the PIs of the PROCESS that created the current PROCESS.

SDL can also route the SIGNALS using the VIA method. In this case, we can specify the channel that we want to use to send the SIGNAL to its final destination. Thus, it is possible to broadcast a SIGNAL with VIA ALL.

Each SIGNAL is processed and discarded following the arrival order. All of the SIGNALS are queued in a FIFO queue that belongs to the PROCESS, as shown in Figure 14. In SDL-2010, the order can be modified by using the delay parameter, as discussed in section 2.3.8.

The SIGNALS in SDL can also include passing parameters. As an example (see Figure 15), a SIGNAL A sends two parameters (True, 5). The PROCESS that receives the signal P2 contains a declaration of two variables, V1 of type Boolean and V2 of type Integer. When the SIGNAL A is received, the INPUT elements of these two variables are filled with the values sent by PROCESS P1.

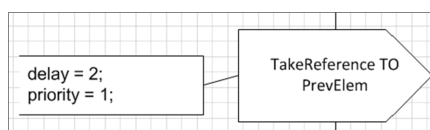
If no variables are defined in the INPUT element of the P2 process, then the values are discarded; it is important to be certain that the types are compatible with the values that are received. It is also possible to ignore some values by simply not including them. As an example, we can define “INPUT A(V2).” In this case, the first value, the Boolean, is discarded.

Because the management of the delays in the simulation model is a key aspect of the model, the manner in which SDL manages delays will be reviewed in the following section.

Working with Time in SDL

In a discrete simulator, to completely define the behavior of a model, it is necessary to describe the time connected to the execution of each of the different events involved in its functions. Usually, each type of event has a specific probability distribution

Figure 16. A delayable SDL signal. Note that this signal needs 2 units of time to reach its destination. Furthermore, the priority is defined to avoid ambiguity that exists if two signals with the same execution time reach the destination at the same time.



that decides when the event must be executed. In an event-scheduling simulator, the engine manages the timing of all of the events and decides where and when all of the events must be sent to other simulation elements, or AGENTS, in an SDL model.

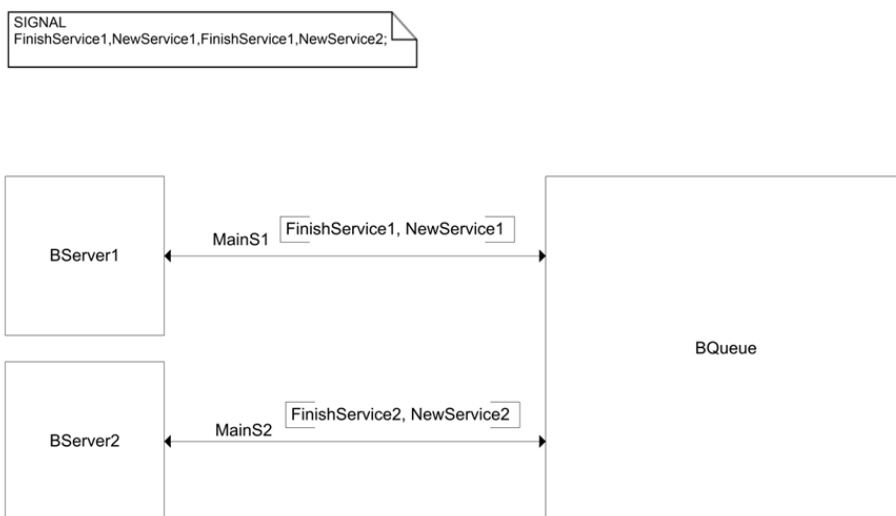
SDL has two main structures for managing time, timers and delaying channels (ITU-T, 1999).

Delaying channels cannot be used as a mechanism for representing the delays in a simulation model because no mechanism exists in SDL to define the time that is required to reach the destination using the channels. The delaying channel represents a delay in the transmission of the signal, but we cannot define the probability distribution that rules the delay. Timers and the other mechanisms that are available in SDL to manage time cannot handle simulation delays because for each different instance of a signal that can travel in parallel, a new timer must be defined. For example, if a signal must be sent to represent the arrival of new entities at a machine, then each time that a new arrival is sent to this machine, the timer is reprogrammed, implying that if the signal has not arrived to its final destination, it can be lost. Only one instance of the signal represented by the timer can travel through the system at a time. Additionally, an important feature of discrete simulation models is that priorities cannot be set using timers.

Figure 17. This diagram shows the main representation of the system. The interaction of the system and the environment is defined at this level. The events that are being sent from the main elements (SDL agents) of the model can also be observed. In this case, the events are FinishService1, FinishService2, NewService1 and NewService2.

system GG2

1 (1)



Specification and Description Language for Discrete Simulation

The problem of how to manage delays in SDL has been studied by several authors (Bozga, Graf, Mounier, Kerbrat, Ober, & Vincent, 2000; (Bozga M., Graf, Mounier, Kerbrat, Ober, & Vincent, 2000) (Bozga M., Graf, Mounier, Ober, Roux, & Vincent, 2001) Bozga, Graf, Mounier, Ober, Roux, & Vincent, 2001). In (Bozga M., Graf, Mounier, Ober, Roux, & Vincent, 2001)Bozga, Graf, Mounier, Ober, Roux, & Vincent (2001), an extension that defines three types of transitions, (i) eager (i.e., consumed without delay), (ii) lazy (i.e., not urgent) and (iii) delayable (has an enabling condition depending on time), are presented. In that study, the authors note that an eager transition is equivalent to a delayable transition with the temporal condition set to $\text{now} = x$ (Bozga M., Graf, Mounier, Ober, Roux, & Vincent, 2001) (Bozga, Graf, Mounier, Ober, Roux, & Vincent, 2001). In general, an event in an event-scheduling simulation engine can carry the following as parameters: (i) ExecutionTime, or delay representing the time when the event must be executed; (ii) Priority, the priority of the event, which is used to avoid a possible simultane-

Figure 18. Decomposition of BlockServer1. In this case, inside the structure of the element, only one PROCESS is defined. The PServer1 PROCESS must contain the description of the behavior of BlockServer1 of the model.

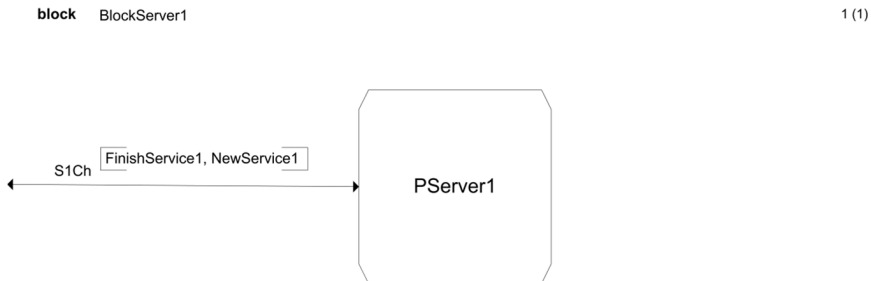
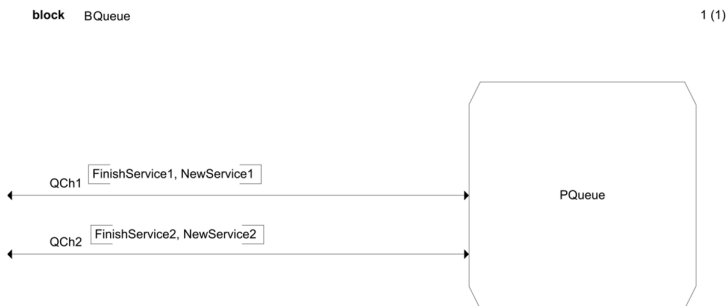


Figure 19. Decomposition of the BQueue. A single PROCESS defines its behavior. Two channels allow communication with both servers of the model.



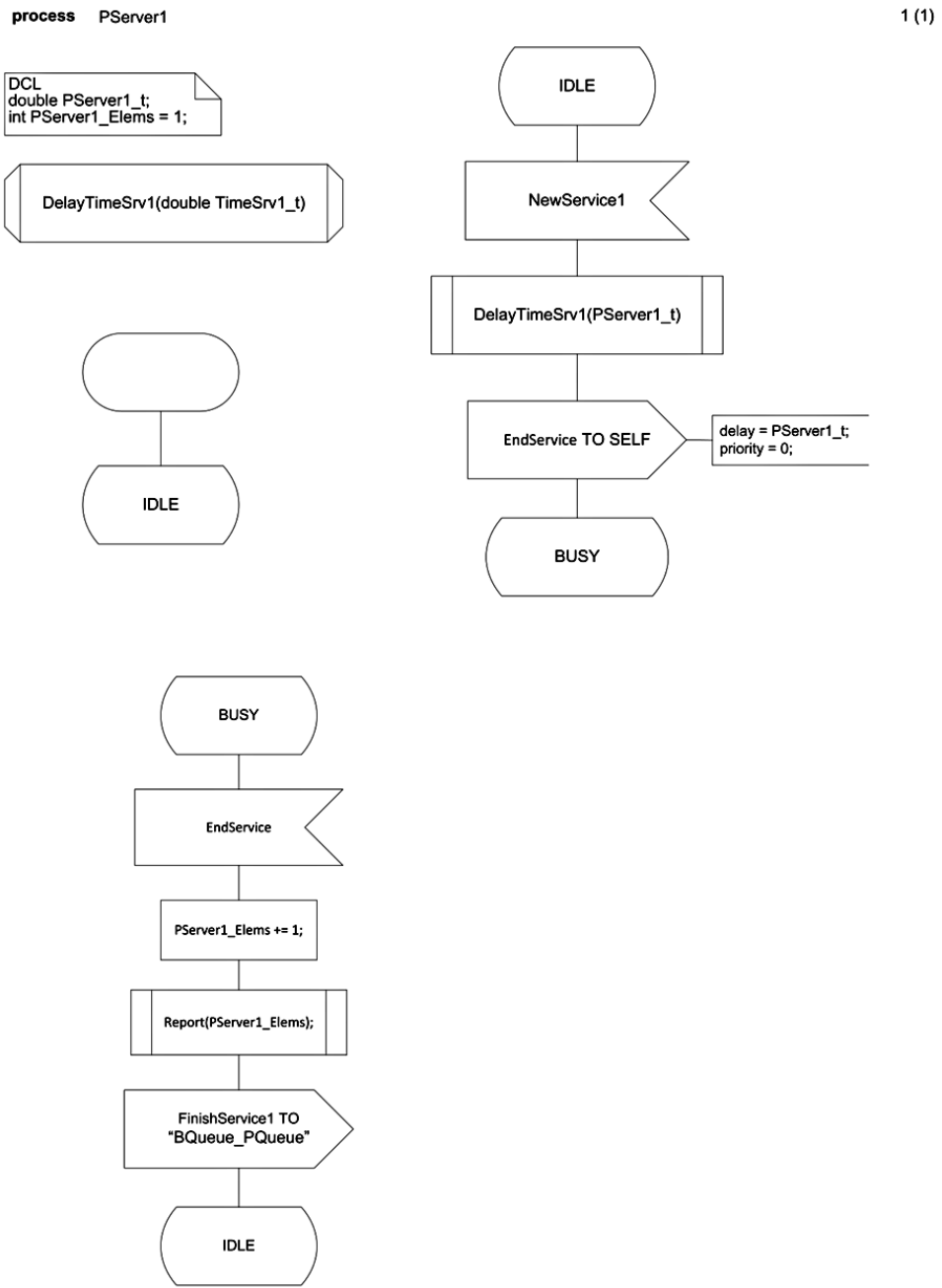
ity of events; (iii) *CreationTime*, representing the time when the event is created; (iv) *Id*, an identifier of the event; and (v) *Time*, the clock of the process (represents the time related to the last event processed by the process). From the point of view of a discrete simulator, all of the transitions can be considered delayable because all of the events have a time that defines when they can be executed. With these considerations in mind, some extensions were proposed (Fonseca i Casas, Colls, & Casanovas, Towards a representation of environmental models using specification and description language, 2010)(Fonseca i Casas, Colls, & Casanovas, 2010; Fonseca i Casas, Colls, Casanovas, & Josep, 2010; (Fonseca i Casas P., Colls, Casanovas, & Josep, 2010) (Casas & Pau, 2010) Casas & Pau, 2010)and finally incorporated into the latest version of SDL (SDL-2010) to simplify the representation of time delays, as follows:

1. All of the signals can have a delay time. In other words, each signal instance output has a parameter that defines the time that is required to travel to its final destination (i.e., a delay or the value of the *ExecutionTime* value minus the current time) and a parameter that defines the priority with respect to other signal instances in the destination input queue scheduled for the same time (i.e., the same *ExecutionTime* value). A signal instance is, therefore, only available

Figure 20. PROCESS PServer1, describing the behavior of the server. Two states are defined in this process, BUSY and IDLE. The first state for the process is IDLE. (The START symbol is connected with the IDLE state.) This situation implies that the PROCESS waits in the IDLE state until it receives a SIGNAL, in this case, the NewService1 signal. Once this SIGNAL is received, the PROCEDURE DelayTimeSrv1 is executed, obtaining a value for the PServer1_1 variable. This value is used to represent the delay of the EndService SIGNAL that is sent from the PROCESS to itself. As seen in the diagram, once the EndService SIGNAL is sent, the PROCESS waits in the BUSY state, representing in the model that the system machine is performing some action with the entity. In the BUSY state, the PROCESS receives the EndService SIGNAL (previously sent by itself with a delay calculated by the DelayTimeSrv1 PROCEDURE). This SIGNAL causes the PROCESS to increment the value of the number of elements served (PServer1_Elems) and generates a report with this information (report PROCEDURE that is not defined in this example). To inform the queue that the server is again IDLE, a SIGNAL (FinishService1) is sent to the queue with no delay, and the PROCESS changes its state to IDLE.

continued on following page

Figure 20. Continued



in the destination input port when the current time is greater than or equal to the ExecutionTime value.

2. The signals in the input port are scanned in a set order (first by ExecutionTime and then by the order of arrival time) to determine whether there is a signal that is enabled. For inputs with the same ExecutionTime and the same arrival time, the signal priority determines which signal is processed first. In the case when two signals have the same ExecutionTime and the same priority, the implementation must define what signal must be executed first (because, in that case, the model does not specify the order).

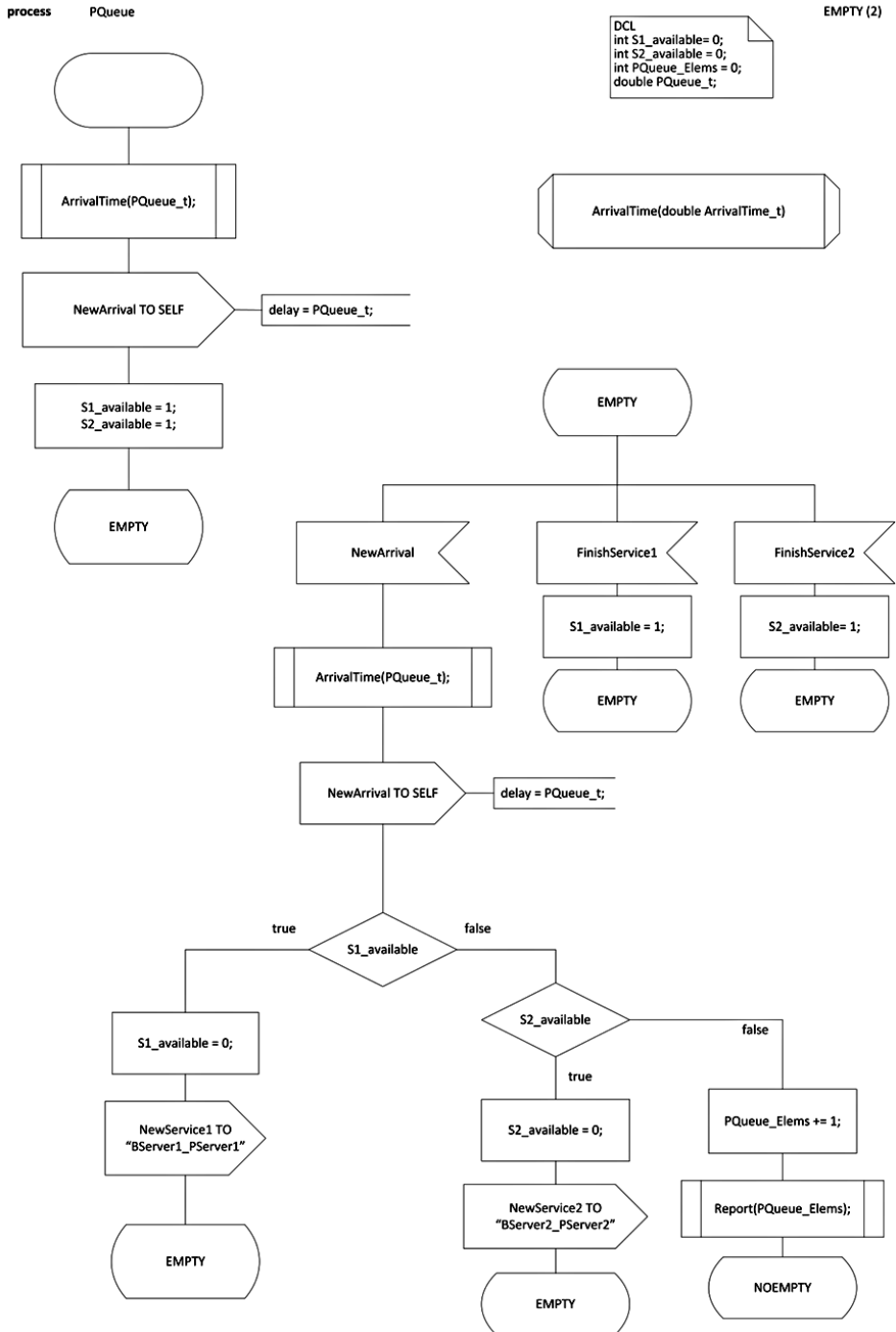
In this chapter, the convenient solution shown in Figure 16 is adopted to represent time and priorities in the SIGNALS.

With all of the system parts defined, a simple system composed by a FIFO queue with two identical servers can now be modeled.

Figure 21. This PROCESS defines the behavior of the Queue. The behavior for the EMPTY state is described. The PROCESS starts by defining the time of the first arrival (in the ArrivalTime PROCEDURE, using the PQUEUE_t variable). A SIGNAL (NewArrival) is sent to itself to represent the arrival of the first client to the system. In the initialization step, two variables, S1_available and S2_available (which represent the states of the servers), are initialized to 1, showing that both servers are IDLE. On an EMPTY state, the queue can receive three different signals, NewArrival, FinishService1 and FinishService2. If the PROCESS receives FinishService1 (signifying that the server has finished its operation), the QUEUE modifies the value that represents the state of the server S1_available to 1 (the server is now free) and remains in the EMPTY state (this procedure is the same for FinishService2). If NewArrival is received by the PROCESS, a NewArrival SIGNAL must be generated (this is usual in every event scheduling entity generation process). Once this procedure is finished, an available server must be found. The first server is inspected first (which shows an inherent prioritization for the first server). If Server1 is available, then we modify the variable showing that it is now BUSY, and we send an immediate SIGNAL to server1. Because the QUEUE does not have elements waiting, it remains in the EMPTY state. If the first server is BUSY, but the second is IDLE (S2_available DECISION), then the process is equivalent (now both servers are working. If both servers are BUSY, then the QUEUE must increment the number of elements waiting (PQueue_Elems +=1;), send or write a report, and change its state to NOEMPTY.

continued on following page

Figure 21. Continued



A SIMPLE GG2 EXAMPLE

In this section, an example of an SDL model that represents a simple queuing system is developed. In this representation, ANSIC is used to represent the code that appears in PROCESSES and PROCEDURES, as suggested in SDL-RT (PragmaDev SARL, 2006), because this choice often creates a model that more closely resembles the code used in the final implementation of the model. Our GIGI24 model, following Kendall's notation (Kendall, 1953), is a representation of a system that is composed of two machines (identical machines) preceded by a queue (FIFO). The entities arrive at the system following a general distribution. If one of the two identical servers is free, it starts its operation with the entity (the time needed to complete the operation is also a general distribution). Once this operation is completed, the entity releases the server (the server is now free and other entities in the queue can enter the server).

The SDL model representation of this system starts with the SYSTEM diagram. This diagram shows the main system elements and the interaction between them and with the environment. In this case, there is no interaction with the environment because the model is self-contained.

The entities are processed by the machines. The machines require time to process the entities. This time, which is the delay of the operation, is represented by a well-known distribution, such as an exponential or Poisson distribution. Once the

Figure 22. This PROCESS defines the behavior of the QUEUE agent. The NO-EMPTY state is described in this agent. In this state, the PROCESS can receive the FinishService1 and FinishService2 SIGNALS (showing that the servers have finished the operation with the entity) or a NewArrival SIGNAL, announcing that a new entity has arrived at the system (with both servers working and some elements in the queue). In this last case, the queue must increment the number of elements in the queue ($PQueue_Elems += 1$);, write or send a report (if desired) and, as usual in an event scheduling generation process, generate the next arrival (calculating the next arrival time and sending the SIGNAL NewArrival to itself). In the case that the SIGNAL received is FinishService1 (or its equivalent FinishService2 for the second server), the queue must see whether some elements are waiting in the queue. If not, the queue must mark that the server is IDLE and return it to the EMPTY state. Otherwise, if some elements are in the queue, then the queue takes one element (decrements the variable $PQueue_Elems$ by one) and sends a NewService1 SIGNAL to server 1 to start its operation. In that case, the queue does not change its state; however, it could be that no other elements are in the queue. To detect this condition, the condition is added that analyzes whether $PQueue_Elems$ is equal to 0, and the state of the queue changes to EMPTY.

continued on following page

Figure 22. Continued

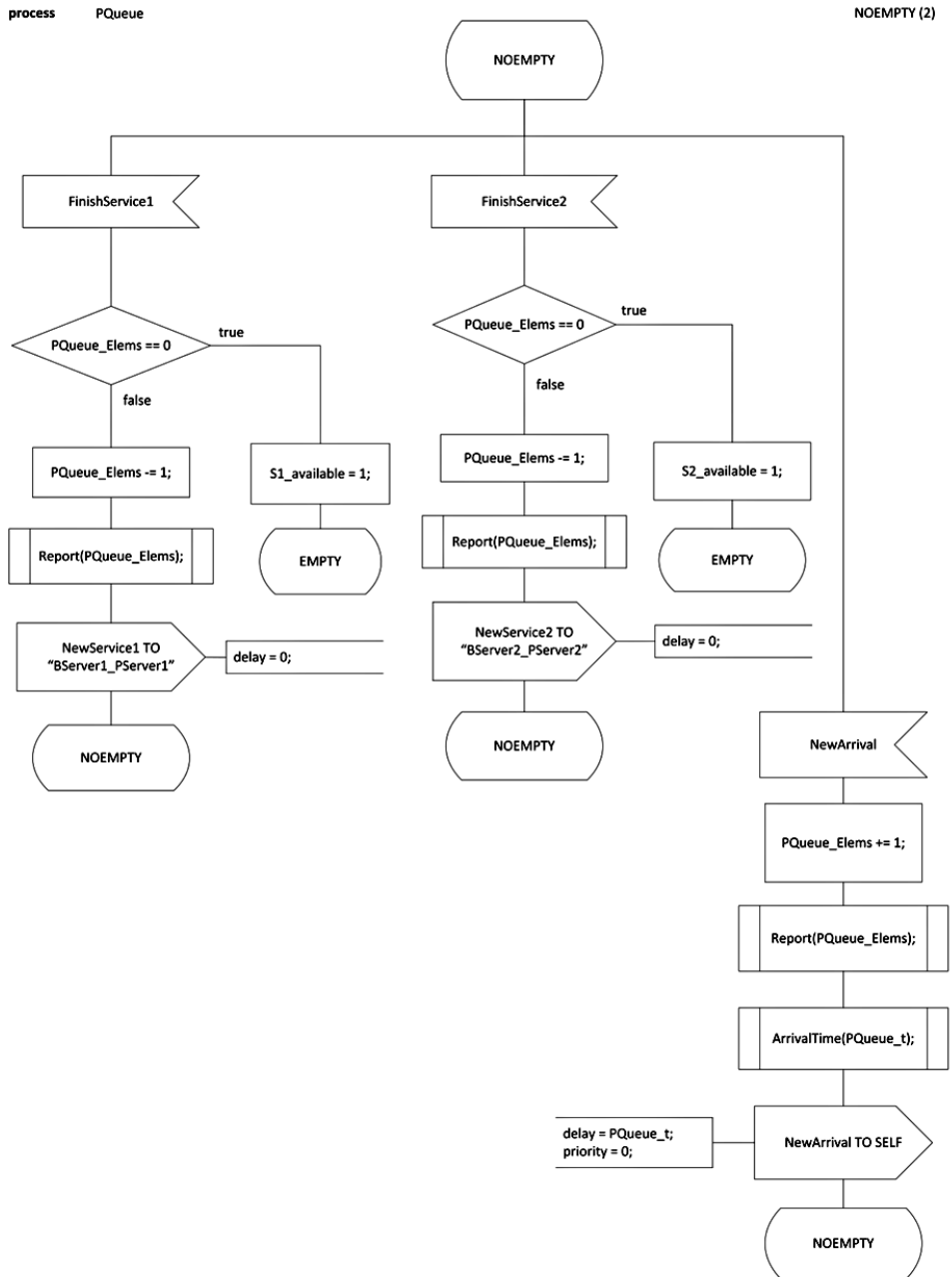
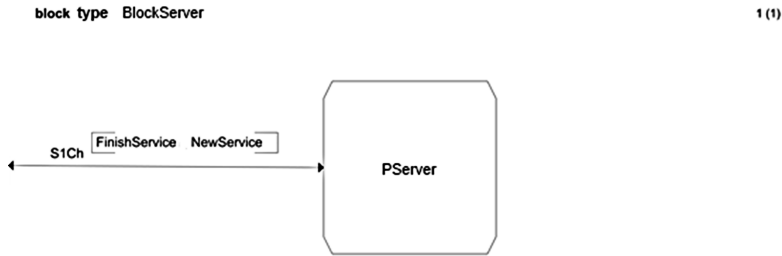


Figure 23. BlockServer type defining the generic behavior of the server

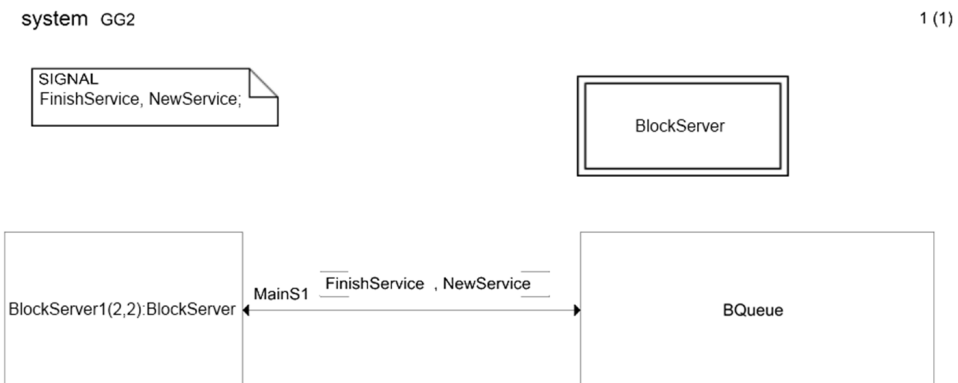


process is finished, the entities leave the system. Figure 17 represents the system that is defined using SDL.

In Figure 17, the agents that can be executed in parallel in the model are shown. In this case, the two servers and the queue can be in different machines. All of the events that are inside a BLOCK agent can be executed in parallel, and all of the events that are inside a PROCESS agent are executed sequentially.

From this initial definition of the model, a hierarchical SDL decomposition can be developed. The elements of BServer1, BServer2 and BQueue (Figure 19) can be

Figure 24. GG2 model using instantiation. The model can be defined by writing the behavior of the different servers once. Both servers are described on BServer(2,2):BlockServer. The first “2” represents the number of instances at the beginning, while the second “2” represents the maximum number of instances of the BlockServer.



Specification and Description Language for Discrete Simulation

Figure 25. PServer PROCESS now can be used by all of the servers in the model. The main difference is that the FinishService SIGNAL now represents the end of the service for all of the servers in the model.

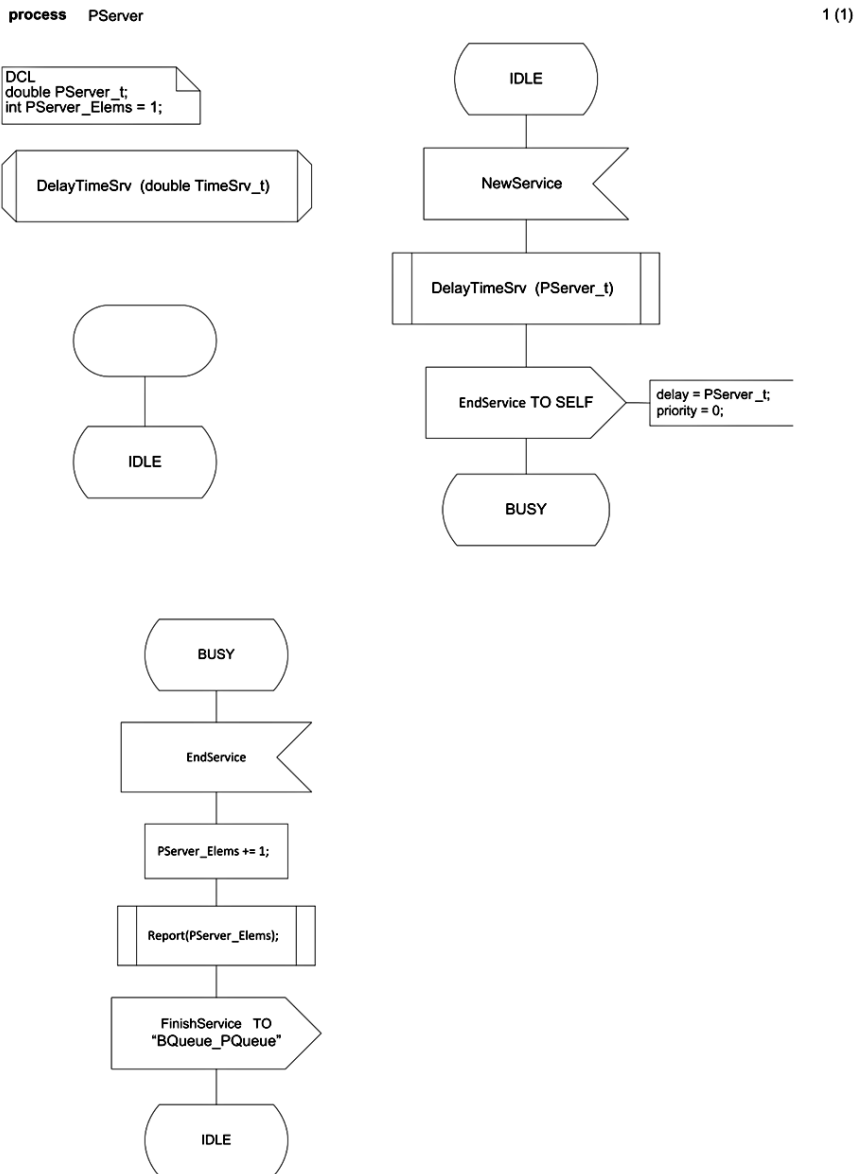


Figure 26. Definition of the PQueue PROCESS for the EMPTY state

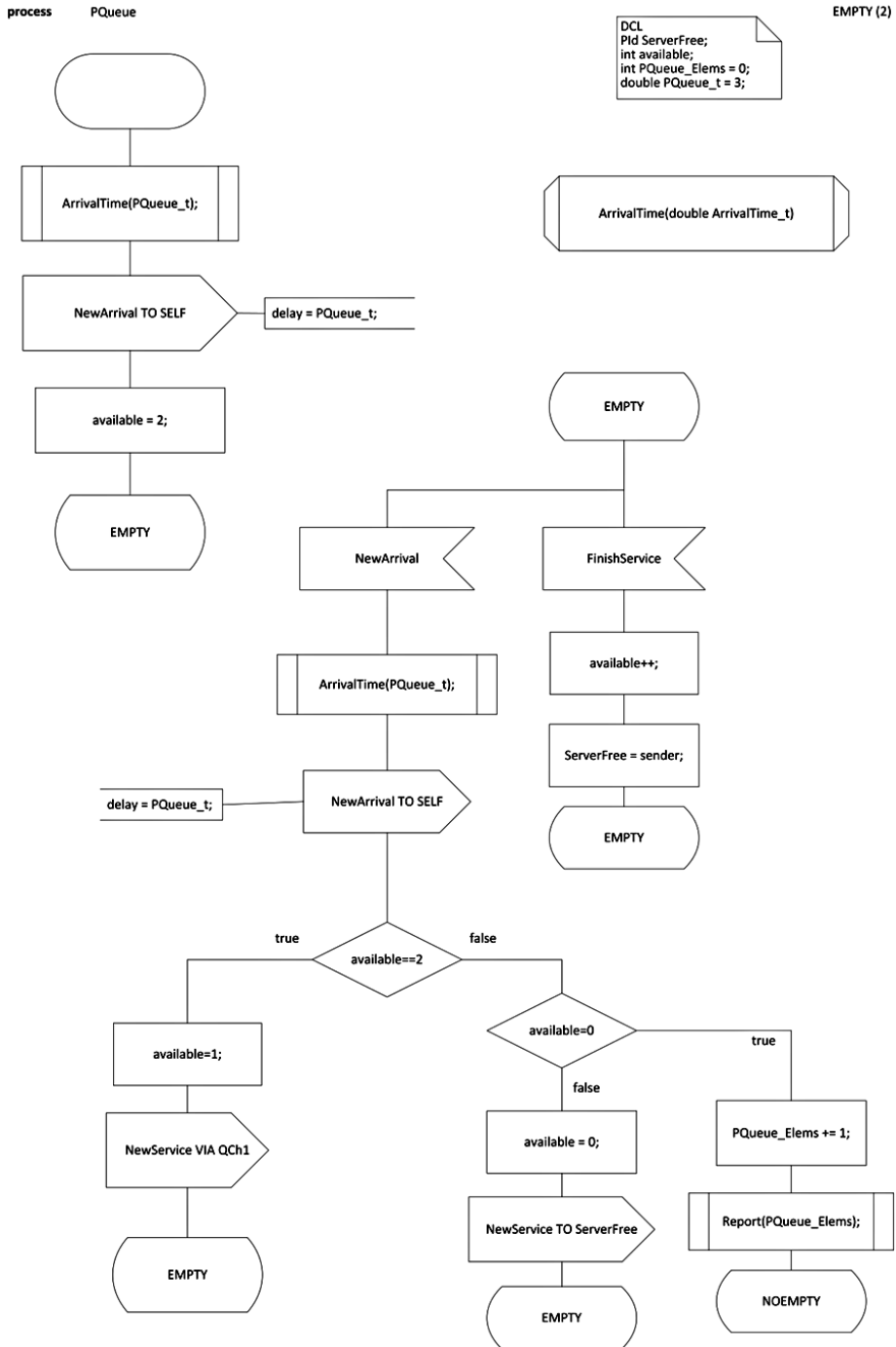
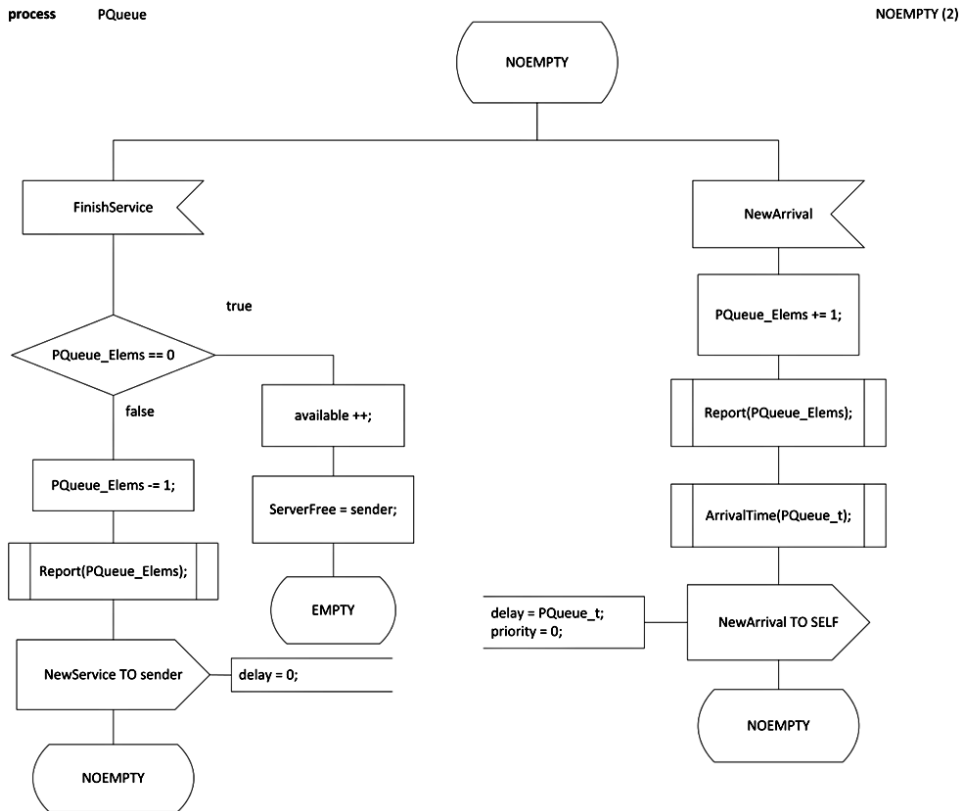


Figure 27. Definition of the PQueue PROCESS for the NOEMPTY state



defined. For each one of these blocks, its complete behavior can be defined based on the definition of new BLOCKS. Finally, some PROCESSES must be defined that detail the specific behavior of the model.

Figure 18 shows the structure of the server. In this case, only one PROCESS is defined.

The channel describes how the SIGNALS (which represent the events in the simulation model) travel through the model. Until this point, the model contained only the description of the structure of the model. The diagrams that define the behavior of the model begin with the PROCESS diagrams. Figure 20 shows the process that defines the behavior of the first server (PServer1). Both of the servers are equivalent. SDL allows the definition of objects and, through inheritance, the definition of instances; in section 3.1, the method SDL uses to manage inheritance

is described. Using inheritance, one definition of a server can be used for all of the identical servers in the model.

In the PROCESS shown in Figure 20, two states are defined, BUSY and IDLE. All of the PROCESS diagrams in SDL begin with the START symbol, “Ò.” This symbol defines the initial transitions that must be executed in each of the model PROCESSES and, hence, the initial conditions for each one of the different PROCESSES. As seen in the diagram, the server starts its execution in the IDLE state. From the point of view of the discrete simulation, between the START symbol and the first state (IDLE in this case), the events (SDL SIGNALS) that initialize the model are sent. As shown in the PROCESS of the queue, the events that are sent represent the generation of the first model elements (clients that are going to be served by one of the two servers). Once the process is in one STATE, it remains there until it receives a SIGNAL. Once it receives a SIGNAL (representing a model event), the PROCESS starts its execution, finishing in a new STATE (which can be the same STATE). The report PROCEDURE represents something that is performed with the information from the model (such as record the information in a file or show a chart); it does not affect the model behavior, but it is necessary to obtain information from the model. Depending on the tool used to code the model, this procedure might not be used if the reports can be obtained from the tool.

Figure 21 and Figure 22 detail the behavior of the queue AGENT. The initial conditions of the model are defined in Figure 21. A PROCEDURE named Arrival-Time defines the time required between two arrivals. Once this value is obtained, it is stored in the PQueue_t variable (see the declaration in the DCL block on the right side of the figure). This variable is used to delay the reception of the SIGNAL (representing the time needed for the first customer to arrive at the system). The SDL-2010 DELAY SIGNAL parameter is used to accomplish this step. During initialization, the state of the servers is stored in the two variables, S1_available and S2_available; thus, the location to send the new customers is known. This step is required because SDL is a modular language (i.e., the state of the other PROCESSES cannot be accessed), and there is no assurance that the implementation of the model will include a shared memory mechanism that allows direct access to the variables of other AGENTS. The only way to know the state of the other AGENTS is to store this information locally or to send a SIGNAL. In addition, each SDL PROCESS can be executed on a different computer.

This simple example illustrates the use of SDL to model a well-known system. In this example, two identical elements are modeled twice; however, with SDL, a class of AGENTS was defined, allowing the reuse of model components. In the next section, this model is reviewed and defined in a simpler way.

SDL Instantiation and Inheritance

SDL is an object-oriented language, accepting instantiation and inheritance as a standard way of defining common behavior for different elements of the model.

Instantiation allows defining from an AGENT TYPE specific instances that can be used inside the simulation model. Each one of the new instances has a different PId.

Inheritance is a general mechanism that applies to interaction diagrams, to behavior diagrams and to data types (Reed, SDL-2000 from New Millenium Systems, 2000)(Reed, 2000). In data types, it is allowed to add new operations. In PROCESS diagrams, we can add new transitions that can lead to new states. For example, if it is desired for Server 2 to define server breakdown, then the diagram can be extended to show what happens, specifically for Server 2, when the SIGNAL NewBreakdown is received. To represent what parts can be rewritten, SDL uses the word redefined. None of the other parts can be changed. Because an AGENT can inherit from an AGENT that inherits from another AGENT (and so on), the system must determine when no more modifications can be performed. To express this criterion, SDL uses the word finalized, which signifies that the new subclasses cannot be modified. On the header of the AGENT that implements a new instance, it must clearly be indicated that the definition is a type with the keyword inherits.

Similar to in an instantiation, the AGENT that inherits from an agent type obtains a new PId.

With these considerations, it is easy to rewrite the model with a single BLOCK defining both servers and with the same name for the SIGNALS that travel from one AGENT to the other (using the TO parameter, as shown in the example).

First, it is necessary to define a BLOCK class that defines the generic behavior of the server (see Figure 23).

Then, it is easy to rewrite the main diagram showing the similar behavior of both servers (see Figure 24).

If needed, extra functionality can be added to the definition of the BLOCKS BServer1 and BServer2.

The diagrams show that PQueue prioritizes the use of Server 1; thus, if both servers are free, the SIGNAL is always sent to the first server (see Figures 26 and 27). This process is conducted using the VIA parameter (it is not necessary to know the PId of the PROCESS).

When one of the two servers is free, the destination of the SIGNAL must be known. In that case, a PId variable, ServerFree, is used to store the PId of the PROCESS that represents the Server that is currently available. (If both servers are free, then this variable is unnecessary because the elements are sent to the first server).

With the above considerations, the model is rewritten as follows. Both PServer1 and PServer2 are now represented by a single PROCESS named PServer, which is defined inside the BServer BLOCK that inherits from BlockServer (see Figure 25).

Several other alternatives exist to handle different instances of an AGENT type; the most common alternative is to save the different PIDs for the elements where the SIGNALS are to be sent. A common procedure to initialize the values is reviewed in (Reed, Re: SDL-News: Request for Help: Initialisation of Pids, 2000)Reed (2000).

TO LEARN MORE

SDL is a modern language that is under continuous evolution. However, it has standard, clear and concise documentation generated by the ITU-T, which helps in the understanding of the language elements and also helps to implement tools based on this language.

The main documentation of the language can be obtained from the ITU-T website, and other documentation sources exist, such as (Doldi L., 2003) Doldi (2001; 2003), (Doldi L., 2001)ITU-T (1999), (ITU-T, 1999)ITU-TS (2004) or (SDL Tutorial) SDL Tutorial. The best source of current information is located on the Website of the SDL Forum Society at <http://sdl-forum.org/>.

In this section, the SDL language is introduced, and its use in the field of operations research is discussed using an interactive example that represents a simple queuing model. However, SDL is a powerful language that can be used to describe the behavior and the structure of more than interactive models. Some examples of further research can be found in Fischer, Kühnlenz, Ahrens, & Eveslage (2009) (Fischer, Kühnlenz, Ahrens, & Eveslage, 2009), Fonseca i Casas, Colls, Casanovas, & Josep (2010)(Fonseca i Casas P., Colls, Casanovas, & Josep, 2010)), (Fonseca, Colls, & Casanovas, 2011)Fonseca, Colls, & Casanovas (2011), Rodríguez-Cayetano (2011)(Rodríguez-Cayetano, 2011), (Braun, Gotzhein, & Wiebel, 2011)Braun, Gotzhein, & Wiebel (2011), (Kraemer, Slåtten, & Herrmann, 2009) Kraemer, Slåtten, & Herrmann (2009), or (Weigert, et al., 2007)Weigert et al., 2007.

REFERENCES

- Bozga, M., Graf, S., Mounier, L., Kerbrat, A., Ober, I., & Vincent, D. (2000). SDL for Real-Time: What Is Missing? SAM'2000. Grenoble, France.
- Bozga, M., Graf, S., Mounier, L., Ober, I., Roux, J.-L., & Vincent, D. (2001). Timed Extensions for SDL. In Proceedings of SDL-Forum'01. Copenhagen, Denmark: SDL.

Specification and Description Language for Discrete Simulation

Braun, T., Gotzhein, R., & Wiebel, M. (2011). Integration of FlexRay into the SDL-Model-Driven Development Approach. In Kraemer, F., & Herrmann, P. (Eds.), *System Analysis and Modeling: About Models* (pp. 56–71). Berlin, Germany: Springer. doi:10.1007/978-3-642-21652-7_4.

Casas, F. I. (2010). Using Specification and Description Language to define and implement discrete simulation models. In Proceedings of the 2010 Summer Simulation Multiconference. Ottawa, Canada: SCS.

CINDERELLA SOFTWARE. (2007). Cinderella SDL. Retrieved March 31, 2009, from <http://www.cinderella.dk>

Doldi, L. (2001). *Sdl illustrated - Visually design executable models*. TRANSMETH SUD OUEST.

Doldi, L. (2003). *Validation of Communications Systems with SDL: The Art of SDL Simulation and Reachability Analysis*. New York: John Wiley & Sons, Inc. doi:10.1002/0470014156.

Fischer, J., Kühnlenz, F., Ahrens, K., & Eveslage, I. (2009). Model-based Development of Self-organizing Earthquake Early Warning Systems. In I. Troch, & F. Breitenacker (Eds.), *Proceedings of MATHMOD 2009*. Vienna, Austria: ARGESIM.

Fishman, G. S. (2001). *Discrete-Event Simulation: Modeling, Programming and Analysis*. Berlin, Germany: Springer-Verlag. doi:10.1007/978-1-4757-3552-9.

Fonseca, P., Colls, M., & Casanovas, J. (2011). A novel model to predict a slab avalanche configuration using m:n-CAk cellular automata. *Computers, Environment and Urban Systems*, 35(1), 12–24. doi:10.1016/j.compenvurbsys.2010.07.002.

Fonseca i Casas, P., Colls, M., & Casanovas, J. (2010). Towards a representation of environmental models using specification and description language. In Proceedings on the International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management. Valencia, Spain: Springer.

Fonseca i Casas, P., Colls, M., Casanovas, J., & Josep, C. G. (2010). *Representing Fibonacci function through cellular automata using specification and description language*. Ottawa, Canada.

Guasch, A., Piera, M. À., Casanovas, J., & Figueras, J. (2002). *Modelado y simulación*. Barcelona, Spain: Edicions UPC.

IBM. (2009). Retrieved from http://publib.boulder.ibm.com/infocenter/rsdp/v1r0m0/index.jsp?topic=/com.ibm.help.download.sdltcn.doc/topics/sdltcn_download63.html

IBM. (2009). TELELOGIC. Retrieved March 31, 2009, from <http://www.telelogic.com/>

ITU-T. (1999). Specification and Description Language (SDL). Retrieved April 2008, from <http://www.itu.int/ITU-T/studygroups/com17/languages/index.html>

ITU-TS. (1997). *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*. Geneva, Switzerland: ITU-TS.

ITU-TS. (2004). *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*. Geneva, Switzerland: ITU-T.

Kendall, D. (1953). Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain. *Annals of Mathematical Statistics*, 24(3), 338–354. doi:10.1214/aoms/1177728975.

Kraemer, F., Slåtten, V., & Herrmann, P. (2009). Model-Driven Construction of Embedded Applications Based on Reusable Building Blocks – An Example. In Reed, R., Bilgic, A., & Gotzhein, R. (Eds.), *SDL 2009: Design for Motes and Mobiles (Vol. 5719)*, pp. 1–18). Berlin, Germany: Springer. doi:10.1007/978-3-642-04554-7_1.

Law, A. M., & Kelton, W. D. (2000). *Simulation Modeling and Analysis*. New York: McGraw-Hill.

PragmaDev SARL. (2006). *SDL-RT standard V2.2*. Paris: Standard, PragmaDev SARL.

PragmaDev SARL. (2012). Retrieved from <http://www.pragmadev.com/product/codeGeneration.html>

Reed, R. (2000). Re: SDL-News: Request for Help: Initialisation of Pids. Retrieved April 2009, from <http://www.sdl-forum.org/Archives/SDL/0032.html>

Reed, R. (2000). SDL-2000 form New Millenium Systems. *Teletronikk* 4.2000, 20-35.

Reed, R. (2000). SDL-2000 new presentation. Retrieved June 18, 2012, from <http://www.sdl-forum.org/sdl2000present/index.htm>

Rodríguez-Cayetano, M. (2011). Design and Development of a CPU Scheduler Simulator for Educational Purposes Using SDL. In *System Analysis and Modeling: About Models (Vol. 6598)*, pp. 72–90). Oslo, Norway: Springer. doi:10.1007/978-3-642-21652-7_5.

Rudolph, E., Grabowski, J., & Graubmann, P. (1999). Towards a Harmonization of UML-Sequence Diagrams and MSC. In Dssouli, R., Bochmann, G. V., & Lahav, Y. (Eds.), *SDL'99 - The Next Millenium*. Bochum, Germany: Elsevier. doi:10.1016/B978-044450228-5/50014-X.

Tutorial, S. D. L. (n.d.). IEC International Engineering Consortium. Retrieved January 2009, from <http://www.iec.org/online/tutorials/sdl/>

Weigert, T., Weil, F., Marth, K., Baker, P., Jervis, C., & Dietz, P. ... Mastenbrook, B. (2007). Experiences in Deploying Model-Driven Engineering. In E. Gaudin, E. Najm, & R. Reed (Eds.), *SDL 2007: Design for Dependable Systems* (Vol. 4745, pp. 35-53). Berlin, Germany: Springer. doi: doi:10.1007/978-3-540-74984-4_3.

KEY WORDS AND DEFINITIONS

AGENT: The term agent is used to denote a system, block or process that contains one or more extended finite state machines.

BLOCK: A block is an agent that contains one or more concurrent blocks or processes and can also contain an extended finite state machine that owns and handles data within the block.

BODY: A body is a state machine graph of an agent, procedure, composite state, or operation.

CHANNEL: A channel is a communication path between agents.

ENVIRONMENT: The environment of the system is everything in the surroundings that communicates with the system in an SDL-like manner.

GATE: A gate represents a connection point for communication with an agent type. When the type is instantiated, it determines the connection of the agent instance with other instances.

INSTANCE: An instance is an object created when a type is instantiated.

OBJECT: The term object is used for data items that are references to values. ITU-T Rec. Z.100 (11/2007) 5.

PId: The term PId is used for data items that are references to agents.

PROCEDURE: A procedure is an encapsulation of part of the behavior of an agent, which is defined in one place but can be called from several places within the agent. Other agents can call a remote procedure.

PROCESS: A process is an agent that contains an extended finite state machine and can contain other processes.

SIGNAL: The primary means of communication is by signals that are output by the sending agent and input by the receiving agent.

SORT: A sort is a set of data items that have common properties.

STATE: An extended finite state machine of an agent is in a state if it is waiting for a stimulus.

STIMULUS: A stimulus is an event that can cause an agent that is in a state to enter a transition.

SYSTEM: A system is the outermost agent that communicates with the environment.

TIMER: A timer is an object owned by an agent that causes a timer signal stimulus to occur at a specified time.

TRANSITION: A transition is a sequence of actions an agent performs until it enters a state.

TYPE: A type is a definition that can be used for the creation of instances and can also be inherited and specialized to form other types. A parameterized type is a type that has parameters. When these parameters are given different actual parameters, different un-parameterized types are defined that, when instantiated, give instances with different properties.

VALUE: The term value is used for the class of data that is accessed directly. Values can be freely passed between agents.

ENDNOTES

1. The model clock is a real number that represents the time in the model. In a continuous paradigm, this number is continuously modified. In a discrete paradigm, the number is modified based on the time between the events that are processed by the simulation loop (Guasch, Piera, Casanovas, & Figueras, 2002).
2. This diagram is used to represent the behavior of a global system to analyze the evolution of events over time. It is usually used to represent the event evolution in an event-scheduling paradigm.
3. Some well-known SDL modeling tools are Telelogic Tau, PragmaDev RTDS, Cinderella, Safire-SDL, and ObjectGeode (now off the market). PragmaDev RTDS supports both SDL and SDL-RT. Some interesting additional projects are JADE, which is a Java-based specification environment, and SDLPS, which is a C++ simulation environment.
4. We are following here Kendall's notation for queuing models, A/B/C/K/N/D, where A represents the arrival process, B the service time distribution, C the number of servers, K the number of places in the system, N the population and D the queue's discipline.