Hybrid System Modeling using the SIMANLib and ARENALib Modelica Libraries

Victorino Sanz [a,1,*], Alfonso Urquia [a], Sebastian Dormido [a]

[a] *Dpto. Informática y Automática, UNED, Juan del Rosal 16, 28040 Madrid, Spain*

François E. Cellier [b]

[b] *Dept. Computer Science, ETH Zurich, CH-8092 Zurich, Switzerland*

Abstract

The ARENALib and SIMANLib Modelica libraries replicate the basic functionality of the Arena simulation environment and the SIMAN language. These libraries facilitate describing discrete-event models using the Arena modeling methodology. ARENALib and SIMANLib models can be combined with other Modelica models in order to describe complex hybrid systems (i.e., combined continuous-time and discrete-event systems). The implementation and design of SIMANLib and ARENALib is discussed. The ARENALib components have been built in a modular fashion using SIMANLib. The SIMANLib components have been described as Parallel DEVS models and implemented using DEVSLib, a Modelica library previously developed by the authors to support the Parallel DEVS formalism. The use of Parallel DEVS as underlying mathematical formalism has facilitated the development and maintenance of SIMANLib. The modeling of two hybrid systems is discussed to illustrate the features and use of SIMANLib and ARENALib: firstly, a soaking-pit furnace; secondly, the malaria spread and an emergency hospital. DEVSLib, SIMANLib and ARENALib can be freely downloaded from http://www.euclides.dia.uned.es/

Key words:
Modelica, Object-oriented modeling, Hybrid systems, Arena, SIMAN, Parallel DEVS

* Corresponding author. Tel: +34 91 3989469.
Email addresses: vsanz@dia.uned.es (Victorino Sanz), aurquia@dia.uned.es (Alfonso Urquia), sdormido@dia.uned.es (Sebastian Dormido), francois.cellier@inf.ethz.ch (François E. Cellier).

[1] This work has been supported by the Spanish CICYT under DPI2007-61068 grant.

1 Introduction

The Arena simulation environment uses a flowchart-based modeling methodology that facilitates the description of discrete-event systems (Kelton et al., 2007; Law, 2007). Systems are described using Arena from the point of view of the entities that flow through them using the available resources. Arena models are structured in a hierarchical and modular way. They are defined by means of a flowchart diagram and static data.

The flowchart diagram is composed by instantiating and connecting predefined components named flowchart modules. Each flowchart module has an interface and an internal behavior. The interface is used to connect with other modules, thus describing the path for the entities. The internal behavior describes the actions performed by the entities while in the component, e.g., delay a certain amount of time, seize and release resources, and record statistics. The simulation results are usually presented in the form of statistical indicators that are calculated during the simulation.

The static data allow to specify component characteristics, such as for instance, the characteristics of the entity arrival processes, resources and queues. The static data are described using the data modules provided to this end by Arena.

The Arena flowchart and data modules are arranged into panels. The main Arena panel is named BasicProcess. This panel includes the Create, Dispose, Process, Decide, Batch, Separate, Assign and Record flowchart modules, and the Entity, Queue, Resource, Variable, Schedule and Set data modules (Kelton et al., 2007). Other Arena panels are AdvanceProcess, AdvanceTransfer, AgentUtil, FlowProcess and Packaging (Kelton et al., 2007).

Arena is based on the SIMAN simulation language (Pegden et al., 1995). Arena modules are high-level constructs whose functionality is equivalent to sets of SIMAN blocks and elements. Arena predefined modules are internally built using SIMAN blocks and elements, which represent lower-level actions. SIMAN and Arena components can be combined in the same model, provided that they have compatible interfaces and manage the same types of information. Arena provides two panels, named Blocks and Elements, that correspond to the components of the SIMAN language.

Arena provides limited support to the description of continuous-time models (Kelton et al., 2007). The linear models whose state-variable derivatives remain constant between discrete events can be described using the Levels and Rates SIMAN blocks, following the System Dynamics approach (Forrester, 1969). A general-purpose programming language (e.g., C, FORTRAN and Visual Basic) needs to be used to describe other types of continuous-time models

and their connection to the Arena discrete-event model.

Other tools support different approaches to hybrid system modeling. For instance, Anylogic facilitates describing continuous-time models using System Dynamics (Forrester, 1969), similarly to Arena. Ptolemy II provides functionality to describe the continuous-time model using block diagrams, similarly to Matlab/Simulink. Based on Ptolemy II, the Building Controls Virtual Test Bed (BCVTB) is a software environment that allows the co-simulation between different simulation programs, including the Modelica modeling environment Dymola among others (Wetter, 2011).

The general-purpose, object-oriented modeling languages support the physical modeling paradigm (Åström et al., 1998). In particular, the Modelica language (Modelica Association, 2012) facilitates the object-oriented description of DAE-hybrid models, i.e., models composed of differential and algebraic equations, and discrete-time events. Modelica supports a declarative description of the continuous-time part of the model (i.e., equation-oriented modeling) and provides language expressions for describing discrete-time events. These features have facilitated the development of Modelica libraries supporting several modeling formalisms and describing phenomena in different physical domains (Modelica Libraries, 2012). Modelica facilitates the reuse of models and model components, which contribute to reduce the cost of new model development (Robinson et al., 2004).

A number of Modelica libraries have been developed for supporting discrete-event modeling formalisms, including StateCharts (Ferreira and de Oliveira, 1999), state graphs (Otter et al., 2005), hybrid automata (Pulecchi and Casella, 2008), Petri Nets (Mosterman et al., 1998), extended Petri Nets (Fabricius and Badreddin, 2002) and Parallel DEVS (Sanz et al., 2010). The description of operation management models using Modelica is analyzed in Mikler and Engelson (2003) and the model of an inventory system is presented.

The Arena modeling methodology is supported by the SIMANLib and ARENALib Modelica libraries (Sanz, 2010). These libraries facilitate the description of discrete-event logistic models in Modelica, such as manufacturing and packaging processes, supply chains, health care systems, and transport and distribution networks among others. The physical modeling paradigm supported by Modelica can be combined with the Arena modeling methodology in order to describe complex hybrid systems. This combination is not currently supported by other modeling and simulation environments. SIMANLib and ARENALib include components to interface with other Modelica models. These components facilitate connecting models composed using SIMANLib and ARENALib to Modelica models developed using other methodologies, e.g., continuous-time models described using the physical modeling paradigm.

The implementation of SIMANLib and ARENALib, and their use for hybrid system modeling are discussed in this manuscript. Remarks on the SIMANLib and ARENALib implementation are provided in Section 2. The architecture and most relevant features of SIMANLib and ARENALib are described in Sections 3 and 4, respectively. The SIMANLib and ARENALib components for interfacing with models developed using other Modelica libraries are described in Section 5. Finally, two case studies are used to illustrate the SIMANLib and ARENALib capabilities for hybrid system modeling. The modeling of a soaking pit furnace and an emergency hospital are discussed in Sections 6 and 7, respectively. These case studies illustrate the capabilities for hybrid system modeling included in the libraries (i.e., external processes and generation of entities on demand).

2 Remarks on the SIMANLib and ARENALib Implementation

The behavior of the SIMANLib components has been formally described in terms of atomic Parallel DEVS models (Zeigler et al., 2000). The use of Parallel DEVS as a base to describe the behavior of SIMANLib components has facilitated the development, maintenance and reuse of the models. The formal specification of the SIMANLib components in Parallel DEVS can be found in Sanz (2010). The SIMANLib components have been developed using DEVSLib (Sanz et al., 2010), a Modelica library that facilitates the description of Parallel DEVS models in Modelica. As discussed in Section 2.2, the use of DEVSLib in the implementation of SIMANLib is based on the similarities between the Parallel DEVS formalism and the Arena modeling methodology.

The SIMANLib library has been used to develop the ARENALib components in a modular fashion. ARENALib components are constructed as a combination of interconnected SIMANLib components. The same structure can be observed in the Arena environment, whose flowchart modules are constructed using the SIMAN language. The behavior of ARENALib components is formally described as coupled Parallel DEVS models, since SIMANLib components are described as atomic Parallel DEVS models.

2.1 Simulation of Hybrid Models in Modelica

The Modelica libraries and models presented in this manuscript have been edited and simulated using Dymola 6.1 (Dynasim AB, 2006). This state-of-the-art modeling environment automatically translates the object-oriented description of the Modelica model into executable code. The model manipulations performed by Dymola include model flattening, index reduction (if

required), solving and sorting of the equations, and tearing of the algebraic loops. A detailed description of these manipulations can be found in Cellier and Kofman (2006).

The simulation algorithm implemented by Dymola has been conceived for simulating DAE-hybrid models. The simulation algorithm is basically as follows (Cellier and Kofman, 2006):

(1) The continuous-time part is solved using a numerical integration algorithm. Dymola 6.1 allows the user to choose between several supported algorithms that include Lsodar, Dassl, Euler, Rkfix (order 2, 3 and 4), Radau IIa (order 5 stiff), Esdirk (order 3, 4 and 5 stiff), Dopri (order 5 and 8), Sdirk (order 4 stiff) and Cerk (order 3, 4 and 5).
(2) If any of the event conditions is met during integration, the integration algorithm is halted and the event instant is determined.
(3) At the event instant, the set of algebraic and discrete equations are solved.
(4) Once the event has been treated, the event conditions are checked again. If a new event is triggered, it is immediately executed. Otherwise, the integration is restarted.

2.2 Application of DEVSLib for developing ARENALib and SIMANLib

Parallel DEVS and the Arena modeling methodology have some characteristics in common, including the following:

- Both facilitate the modular and hierarchical model description.
- Each model component has an internal description and an interface that is composed of input and output ports. The interaction among the model components is described by connecting their ports.
- The interaction consists in the exchange of information at the event instants. The information sent by an output port is instantaneously received by the input port connected to it. This information is named *message* in Parallel DEVS and *entity* in Arena.
- Components have discrete-event behavior. The component state changes only at event instants.

A discussion on the requirements needed to describe Parallel DEVS models using equation-based, object-oriented modeling languages in general, and Modelica in particular, is provided in Sanz et al. (2010). The communication mechanism among model components is the main difference between Parallel DEVS and the DAE-hybrid modeling formalism supported by Modelica. Model communication in Parallel DEVS follows a message passing mechanism, whereas model connection in Modelica is based on the energy-balance principle.

DEVSLib is a full-fledged Modelica library that facilitates the description of discrete-event models according to Parallel DEVS. DEVSLib supports a message passing mechanism for communicating Parallel DEVS models. Input and output port classes are provided in DEVSLib. The Parallel DEVS model interfaces can be built by defining as many port instances as required. The connection among DEVSLib models is defined by connecting the output ports to the corresponding input ports, using the Modelica *connect* sentences.

DEVSLib allows the user to define the type of information of each message. The default message type contains the following two pieces of information: Type (represented by an integer value) and Value (represented by a real value). The message also includes a Port value that represents the port the message has been received through, but this value is managed by the receiver model and not by the user. As several messages can be simultaneously sent through an output port, this type of message can be used to transmit arbitrarily complex information. In particular, the message passing mechanism supported by DEVSLib has been used for implementing the entity transfer in ARENALib and SIMANLib.

Since SIMANLib and ARENALib are described using the Parallel DEVS formalism, multiple entities can be simultaneously transferred between model components. In this case, the action performed by the component is applied to each received entity at the same time. In components without time delay for the entities, i.e., assignments, statistical indicators, etc., the simultaneous entities will arrive and leave the component at the same time. For components with a time delay, each received entity is inserted in the waiting queue at the same time but their departure will be calculated using the delay time, which is usually random.

The interface of DEVSLib, SIMANLib and ARENALib models is described using the inPort and outPort models included in the DEVSLib library. These interface models can also be used to describe the interface of coupled models. Thus, models can be hierarchically described as a combination of DEVSLib, SIMANLib and ARENALib components.

On the other hand, DEVSLib provides interface components to connect the output of a Parallel DEVS model with the input of a continuous-time model, or vice-versa. Since ARENALib and SIMANLib components are constructed using the DEVSLib library, these interfaces have also been used to combine the models composed using ARENALIb and SIMANLib with other Modelica models. This topic will be addressed in Section 5.

As discussed previously, DEVSLib has been used to describe the discrete-event behavior of the ARENALib and SIMANLib components. Also, the DEVSLib communication mechanism among Parallel DEVS models has served as a basis for implementing the communication (i.e., the entity transfer) among ARENALib/SIMANLib components. Finally, the DEVSLib interface components, intended to communicate the Parallel DEVS models with other Modelica models, have been used for connecting ARENALib/SIMANLib models to other Modelica models. Nevertheless, the development of SIMANLib and ARENALib has required implementing additional functionality. Some SIMANLib/ARENALib features not supported by DEVSLib are discussed below. In particular, those related with the management of the entities and the estimation of the statistical indicators.

According to the Arena modeling methodology, entities flow through the flowchart modules transporting information. A part of this information is common to all entities. Another part contains user-defined pieces of information, named *attributes* in Arena terminology, that are specific to each entity. Models may contain different entity types.

Entities have been implemented in SIMANLib and ARENALib using Modelica records. The entity information is stored in the Modelica record fields. An external library coded in C, named `entities.c`, has been programmed to manage the records. This C library allows to store the records in C structs allocated in dynamic memory, and to read and modify the structs. The message passing mechanism of DEVSLib is used for transferring the entities. To this end, the DEVSLib messages transport in their Value variable a reference to the C struct of the corresponding entities. The resulting mechanism for entity transfer is transparent to the user. The implementation details can be found in Sanz (2010).

A temporal storage for entities is implemented and used in some SIMAN blocks, for instance, in those blocks representing processes that delay the entities. Since the value of the delay time is usually random, the order of the arrived entities could not correspond to the order of the entities leaving the process. These processes have to include a temporal storage for the entities that are being delayed. Some SIMANLib models include an internal queue, similar to the one used to receive messages from other models, that can be used as a temporal storage for delayed entities. Entities in this temporal queue can be ordered depending on their arrival time, or the time they will finish the delay. In the latter case, the first entity in the queue will be the first to leave the process. If multiple entities have the same finishing time, all of them are removed simultaneously from the queue and leave the process.

Simulation results are usually reported using statistical indicators. Some of these statistical indicators have to be calculated during the simulation and some others at the end. The amount of data that has to be stored to calculate some of these indicators changes depending on the length of the simulation. Therefore, a structure to store dynamic information in Modelica needs to be developed, giving the possibility to increase or decrease the size of the stored data during the simulation run. This storage structure has to be accessible from multiple points in the model, in order to facilitate the insertion and removal of data. This behavior is currently prevented in Modelica by the single assignment rule, that forces each variable to be assigned only once in the model (Modelica Association, 2012). An information storage structure, named *dynamic object*, has been developed and used to describe special attributes of the entities, global variables of the model and to store statistical indicators.

A *dynamic object* is a two-dimensional variable (i.e., a matrix) of real type that is stored in dynamic memory. An external library coded in C, named `objects.c`, has been programmed to manage dynamic objects. Dynamic objects are represented in Modelica using an Integer variable that stores a reference to the object in memory. It is similar to the entity record described previously, but in this case the C struct stores a two-dimensional matrix of real numbers and the size of each dimension. The `objects.c` library also supports the description of lists of dynamic objects. These are dynamic lists whose length can be modified during the simulation run, depending on the insertion and removal of objects in the list. Further details are discussed in Sanz (2010).

3 The SIMANLib Modelica Library

The SIMANLib Modelica library provides a subset of the modeling functionality found in the SIMAN language. The top-level architecture of SIMANLib is shown in Fig. 1a. The library is divided in two areas: a user's area and a developer's area. This division helps the user to focus on the components oriented for either developing new models or extending the functionality of the library with new components. The developer's area is encapsulated in the *SRC* package, and contains the developer-oriented documentation and the internal implementation of the components of the library. The user's area is composed of the following top-level classes:

- The *User's Guide*, that includes the user-oriented documentation.
- Components in SIMANLib are divided, as well as in the SIMAN language, in two groups: blocks and elements. The *Blocks* package (see Fig. 1b) contains components to describe the flowchart diagram of the model. The *Elements* package (see Fig. 1c) contains components to specify the static information of the model and the characteristics of its flowchart diagram blocks.

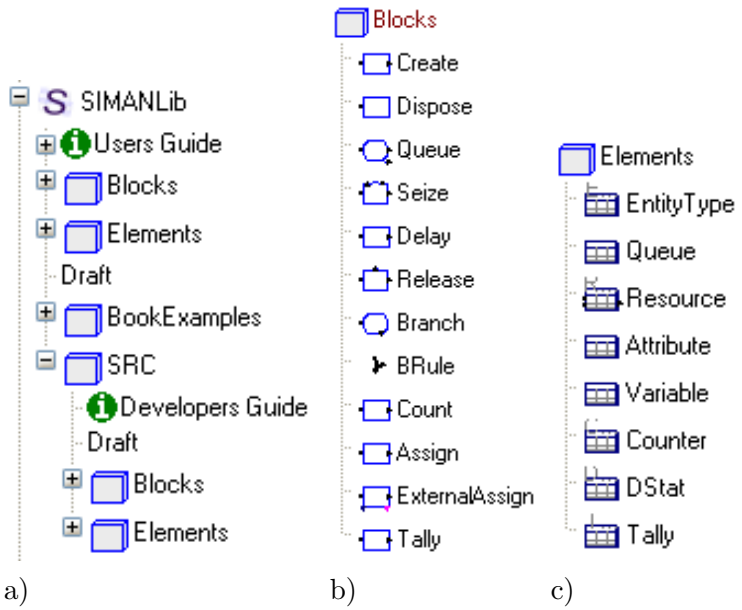


Fig. 1. The SIMANLib Modelica library: a) top-level packages; b) Blocks package; and c) Elements package.

This information corresponds to the entity types, the characteristics of the queues and resources, the global variables, the attributes and the statistical indicators.

- The *Draft* model, that is used as starting point for constructing new models using SIMANLib.
- The *BookExamples* package, that contains several case studies described in Pegden et al. (1995). These examples facilitate the understanding and use of the library. They have been used to validate SIMANLib, comparing them with equivalent models developed using SIMAN/Arena.

The *Elements* package contains classes to represent static information of the model (see Fig. 1c). These elements are implemented using Modelica records and a set of functions to manage the record information. The records contain the variables required to store the element information. Some remarks are given below.

Resource represents the available resources that can be used to process the entities. Each resource is divided into resource units that can be individually seized by the entities. Multiple processes can share the same resources. The Resource element has been implemented as an atomic Parallel DEVS model. It receives, seizes and releases petitions for the represented resources, and sends confirmations for the captured resources. In this way, multiple Seize and Release blocks could be connected to the same Resource element (representing processes that share a resource).

The *Variable* and *Attribute* elements represent global variables and user-defined attributes, respectively. These elements have been designed using dynamic objects. The Modelica record used to represent them includes a variable, named *P*, that contains a reference (a pointer) to the dynamic object used to store the actual value of the element. Two functions, *get* and *set*, are provided to read and modify the value of the dynamic object during the simulation.

The elements used to store statistical indicators (*Counter*, *Tally* and *DStat*) have also been designed using dynamic objects. The *Counter* behaves similarly to the *Variable* and *Attribute* elements, using a dynamic object to store the value of the counter. Since the *Tally* and *DStat* elements automatically calculate the required statistical indicator, they include four dynamic objects to store the maximum, minimum, average and last observed values. At the end of the simulation, the final values for each indicator are written into a text file. The *Tally* and *DStat* elements also include a list of dynamic objects. This list is used to store the observations performed during the simulation, and allow the calculation of the confidence interval for the observed indicator.

4 The ARENALib Modelica Library

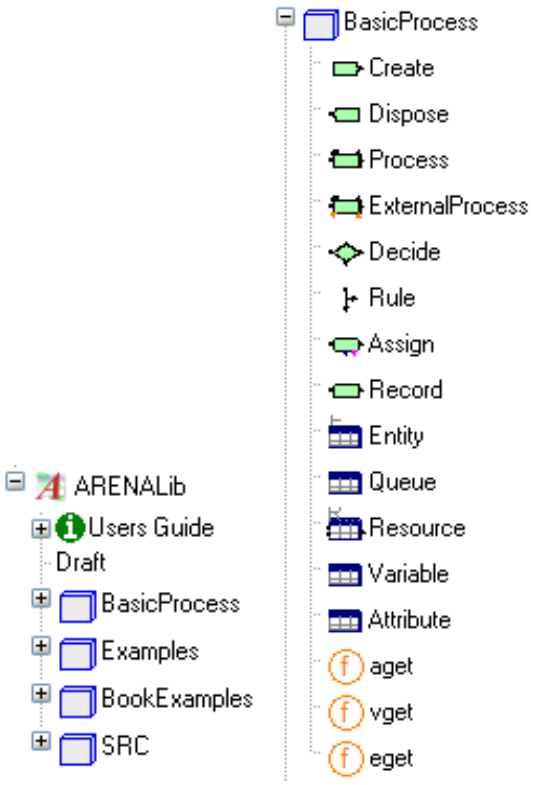
ARENALib reproduces most of the modeling functionality of the Arena *Basic Process* panel. The top-level classes of ARENALib are shown in Fig. 2a. The library is also divided into two areas: a user's area and a developer's area. The developer's area is encapsulated into the *SRC* package and contains the internal implementation of the library modules. The user's area is composed of the following Modelica classes:

- The *User's Guide* model contains the user-oriented documentation of the library.
- The *Draft* model is used to create new models.
- The *BasicProcess* package (see Fig. 2b) contains flowchart and data modules that can be used to construct models.
- The *Examples* package contains several models of discrete-event systems.
- The *BookExamples* package contains models of systems described in Kelton et al. (2007).

ARENALib *flowchart modules* are similar to SIMANLib blocks. However, ARENALib flowchart modules perform more complex actions than SIMANLib blocks and include the calculation of several statistical indicators. ARENALib flowchart modules are described as coupled Parallel DEVS models, and implemented using a combination of SIMANLib blocks and elements. Some characteristics of the ARENALib flowchart modules are discussed below. Their internal structure is shown in Fig. 3.

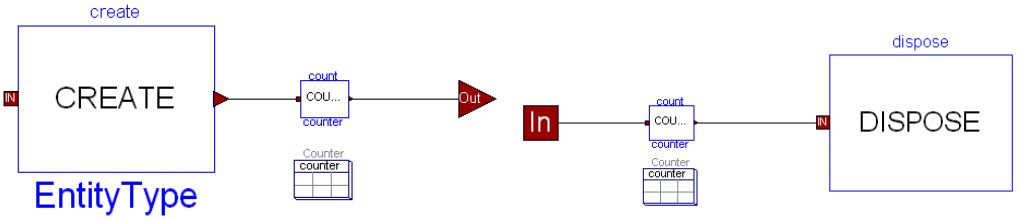
- The *Create* module (see Fig. 3a) represents a source of entities in the system. Entities are created periodically, following the selected inter-arrival time (*Interval* time), and sent through the flowchart diagram. The module has an input port, named IN, to receive external petitions of entity creation. If this input port is connected, the Create module creates entities on demand. New batches of entities are created when a new message is received, instead of every *Interval* time. This module automatically calculates the number of entities created during the simulation run.
- The *Dispose* module represents an end point for entities in the system. Entities are removed (i.e., deleted from the simulation) when they reach this module. It automatically calculates the number of disposed entities.
- The *Process* module represents any action that can be performed by the entities in the system. Processes can be of the following types:
  - *Delay* represents a time delay for the entities, like the Delay block.
  - *Seize-delay* forces the entity to capture a resource before being delayed.
  - *Delay-release* forces a delay for the entity and the release of a previously seized resource.
  - *Seize-delay-release* represents an entity seizing a resource, being delayed and at the end releasing the resource.

  The process type is selected using one of the parameters of the module.

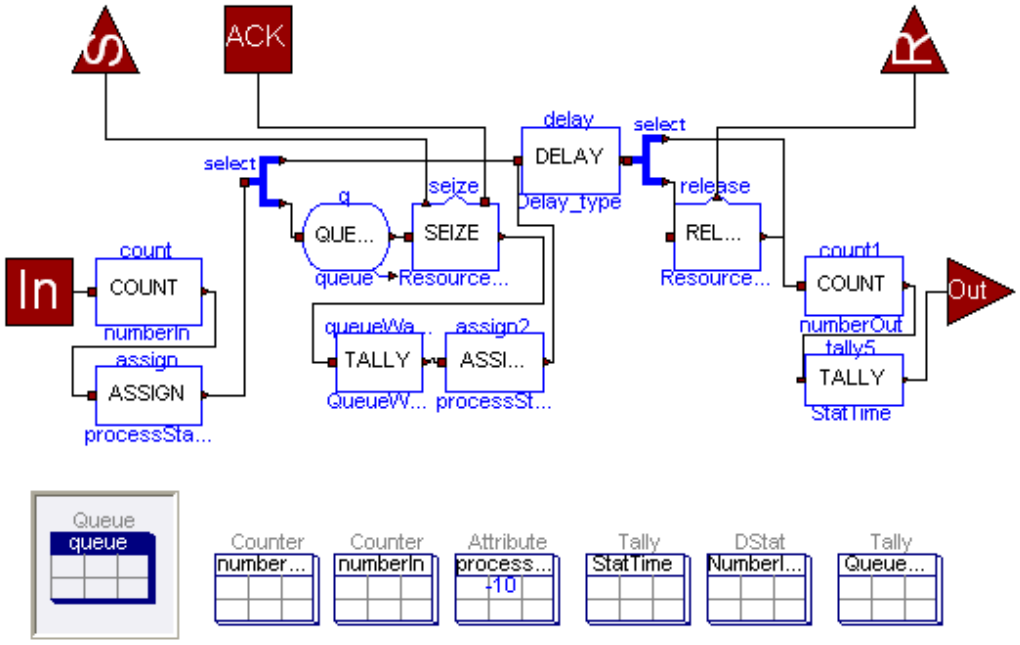


a) b)

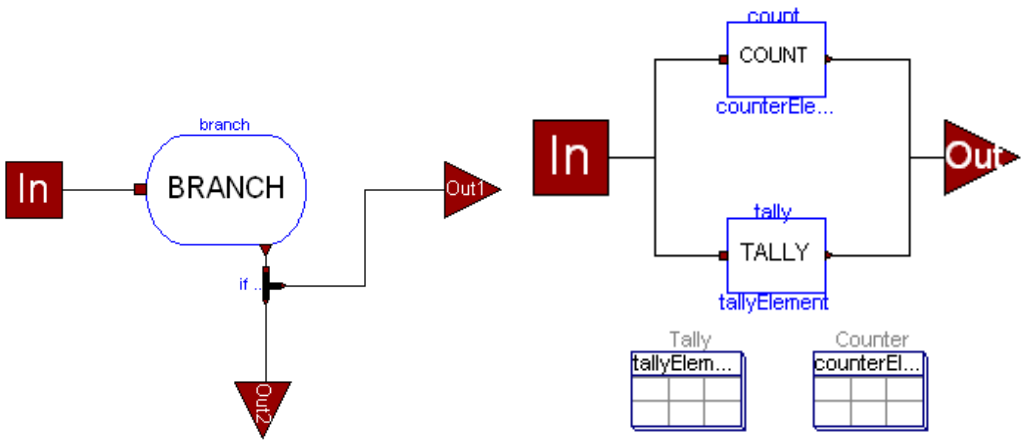
Fig. 2. The ARENALib Modelica library: a) top-level classes; and b) classes within the BasicProcess package.



a) Create b) Dispose



c) Process



d) Decide e) Record

Fig. 3. Internal structure of the ARENALib flowchart modules.

Notice that two Select models, from the DEVSLib library, are used to select the type of process to perform (see Fig. 3c). This module includes the Queue element required for the Seize block, and the elements required to automatically calculate the following statistical indicators:
  – The number of entities that entered and left the module.
  – A Tally indicator for the time the entities are processed.
  – A Tally indicator for the time spent waiting in queue.
  – A DStat indicator for the number of entities waiting in queue.
- The *Decide* module represents a division in the flow of entities following certain conditions or probabilities. It is constructed using a Branch and a BRule block, so only one condition can be checked in the Decide module. In order to allow multiple conditions, ARENALib includes the Rule module, which is equivalent to the BRule block. Additional Rule modules can be connected to the Out2 port of the Decide module.
- The *Record* module represents a point in the flowchart diagram to record statistical time-dependent information. This module is composed of a Tally and a Counter block, which are conditionally declared depending on a parameter, named Type, of the module. If the Type parameter of the module has value 1, the Record behaves as a Counter block and so the Tally block is not declared. On the other hand, if the Type parameter has value 2, the Record behaves as a Tally block and the Counter is not declared. The module also includes the required Tally and Counter elements (see Fig. 3e).

The ARENALib *data modules* are also included in the BasicProcess package (see Fig. 2b). These modules, which are equivalent to some of the SIMANLib elements, facilitate describing certain model static properties. The purpose of the data modules included in ARENALib is discussed below.

- *Entity* is equivalent to the EntityType element in SIMANLib. It describes the characteristics associated with a type of entities in the system. Multiple Entity modules can be included in the same model to represent different types of entities.
- *Queue* is equivalent to the Queue element in SIMANLib. The use of this data module is not required since it is already included in the Process module.
- *Resource* is equivalent to the Resource element in SIMANLib. Each Resource module describes a type of resource in the system, and has to be connected with the Process modules in the model. Several Process modules can be connected to the same Resource, if resource sharing is required.
- *Variable* is equivalent to the Variable element in SIMANLib. It describes user-defined global variables in the model.
- *Attribute* is equivalent to the Attribute element in SIMANLib. It describes user-defined attributes for the entities in the model.

The *BasicProcess* package also includes three functions, named *eget()*, *vget()* and *aget()* (see Fig. 2b), which can be used to read the variable values of an

Entity (eget), a Variable (vget) or an Attribute (aget). In this way, the values of the variables defined in Entities, Variables or Attributes can be used to configure the parameters of the flowchart modules. For instance, an attribute can be assigned with the time of creation for the entity, and its value used as a parameter to calculate the duration of a process or as a condition for a Decide module.

5 Interfacing with other Modelica Models

The Parallel DEVS models described using DEVSLib can communicate with other Modelica models in two ways (Sanz et al., 2010): through direct connections and using the interface models included in DEVSLib.

- *Direct connections*. Variables of other Modelica models can be inputs to the transition functions of the DEVSLib Parallel DEVS models. These connections are similar to the interactions described in the DEV&DESS formalism between the discrete-event and the continuous-time parts of a hybrid model (Zeigler et al., 2000).
- *Interface models*. The communication among DEVSLib Parallel DEVS models takes place exchanging messages. Messages are sent through the output ports and received through the input ports of the Parallel DEVS models. The interface models of DEVSLib can be used to translate messages into discrete-time signals and the other way around, i.e., to translate continuous-time and discrete-time signals into messages. The functionality of the following interface models is explained below: Quantizer, CrossUP, CrossDOWN and DICO.
  - The *signal-to-message interfaces* translate continuous-time signals into event trajectories, where each event corresponds to the transmission of a message. Two different implementations of this interface are included in DEVSLib: quantization (Quantizer model) and value-crossing (CrossUP and CrossDOWN models) interfaces. The quantization interface generates an event (i.e., a message) for every change in the continuous-time signal bigger than a given quantum value. The value-crossing interface generates an event every time the continuous signal crosses a given threshold in one direction, upwards or downwards.
  - The *message-to-signal interface* translates the received message values (i.e., the Value variable of received messages) into a piecewise-constant real signal. A boolean output is also included, together with the real signal output, in order to notify the reception instant of the messages. This boolean output may be useful when the received messages have the same value and consequently the reception instants cannot be inferred from the real signal output. This interface is implemented by the DICO model.

The signal-to-message interface models of DEVSLib can be used to describe the interaction between continuous-time models and the flowchart diagrams composed using SIMANLib/ARENALib. An example is shown in Section 7, where the entity inter-arrival time in the hospital model is calculated from the continuous-time model of the malaria spread. To this end, a Quantizer model is connected to the IN port of the SIMANLib Create block that describes the entity source (infected people arriving to the hospital).

In addition, SIMANLib and ARENALib contain models that describe the interaction between the flowchart diagram and other Modelica models. Two of these interface models are described below: the *ExternalAssign* block of SIMANLib and the *ExternalProcess* module of ARENALib. ARENALib provides a module, named *Assign*, that is equivalent to the ExternalAssign block.

5.1 The ExternalAssign block of SIMANLib

The *ExternalAssign* block has the same functionality as the Assign block: it sets the value of an entity attribute or a model variable. In addition, ExternalAssign communicates the assigned value through its interface. The internal structure of ExternalAssign is shown in Fig. 4. It is composed of a SIMANLib Assign block and three DEVSLib components: DUP, SetValue and DICO. The interface of ExternalAssign is composed of the IN and OUT ports, used to receive and send entities, and the Y (of Real type) and CHANGE (of Boolean type) variables. When the value of an attribute or variable is set in ExternalAssign, this value is also assigned to Y, and CHANGE is switched from true to false or vice-versa. In this way, changes in the value of attributes or variables can be observed by checking the values of Y and CHANGE. The behavior of this block is presented in Listing 1 in the form of an abstract simulator. As shown in the abstract simulator, the actions performed by the block are applied to all the received entities, in the case of simultaneous reception of several entities.

```
when new entities arrive at IN port then
  for each entity in bag loop
    if assignVariable then
      variable := newValue;
    else
      entity.attribute := newValue;
    end if;
    duplicate(entity) to create auxMessage;
    send(entity) to next block through the OUT port;
    auxmessage.Value = newValue;
    y = auxmessage.Value;
    change = not change;
  end for;
end when;
```

Listing 1. Abstract simulator of the *ExternalAssign* block.

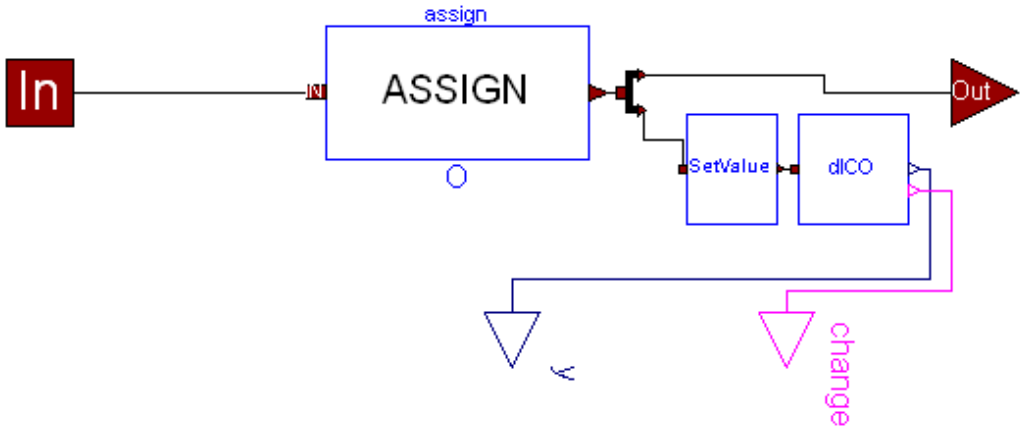


Fig. 4. Internal structure of the SIMANLib ExternalAssign block.

5.2 The ExternalProcess module of ARENALib

The *ExternalProcess* module represents a process whose delay time is externally modeled. The module structure is shown in Fig. 5. ExternalProcess is similar to the Process module (see Fig. 3c), where the Delay block has been replaced by four models (DICO, Quantizer, RealToInteger and IntegerToReal) and two additional interface variables (entityStart and entityEnd). DICO and

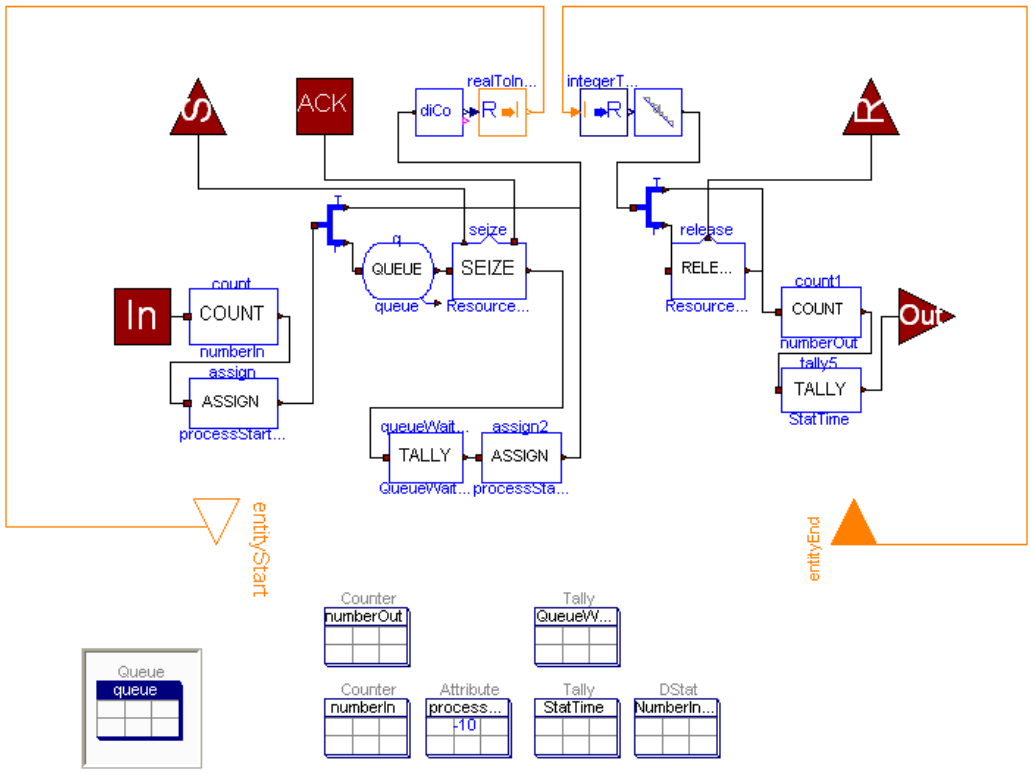


Fig. 5. Internal structure of the ARENALib ExternalProcess module.

Quantizer are defined in DEVSLib, and RealToInteger and IntegerToReal in the Modelica Standard Library. The entityStart and entityEnd interface variables are intended to communicate with the model that describes the delay action.

The DICO and RealToInteger models translate the message representing the entity into an Integer value that is assigned to the entityStart variable. Since this entity identification value is different for each entity (it is the memory address where the entity is stored), the change in the entityStart value means that a new entity is ready to be processed in the delay action. The Modelica model that describes the delay action has to be connected to entityStart and entityEnd. Once the delay time for the entity is elapsed, this model should set the value of the entityEnd variable to the entity identification value. This value is translated by the IntegerToReal and Quantizer models into a message representing the entity, which continues through the flowchart diagram. The behavior of this module is presented in Listing 2 in the form of an abstract simulator.

The management of simultaneous entities is analogous to the ExternalAssign block. Each received entity is either inserted in the queue or its identification value is assigned to the entityStart variable. The latter case will generate several changes at the same time in the value of the entityStart variable, that have to be properly managed by the model that represents the external process.

```
when new entities arrive at IN port then
  for each entity in bag loop
    record statistics;
    if process requires to seize resource then
      insert entity in queue to wait for idle resource;
    else
      entityStart := pointer to entity;
    end if;
  end for;
end when;

when resource idle and queue not empty then
  seize resource for waiting entity;
  remove entity from queue;
  entityStart := pointer to entity;
end when;

when entityEnd value changes then
  recover entity using entityEnd value;
  if process requires to release resource then
    release resource;
  end if;
  record statistics;
  send(entity) to next block through OUT port;
end when;
```

Listing 2. Abstract simulator of the *ExternalProcess* module.

6 Soaking-Pit Furnace

The soaking-pit furnace described in Kelton et al. (2007) is modeled using ARENALib. The use of the Modelica language and the ARENALib Modelica library facilitates the model description. Instead of using Levels and Rates from the SIMAN language to describe the continuous-time equations of the furnace, as performed in Kelton et al. (2007), Modelica allows to directly code these equations (cf. lines 19 to 25 in Listing 3), which will be automatically handled and translated into executable code by the Dymola modeling environment. The ExternalProcess module (see Section 5.2) is employed to interface between the flowchart diagram and the continuous-time model.

The furnace has nine slots for heating ingots. The ingots arrive, one at a time, with exponential inter-arrival time and are positioned, one ingot per slot, in the available slots. If there is no available slot, the ingot must wait in a FIFO queue. When an ingot reaches its optimal temperature, it is removed from the slot, which gets ready to be occupied by another ingot.

The furnace temperature (T) and the temperatures of the ingots placed inside the furnace (τ_i with $i : 1, \ldots, 9$) are described by Eqs. (1) and (2). Temperatures are expressed in degrees Fahrenheit.

$$\frac{dT}{dt} = 2 \cdot (2600 - T) \tag{1}$$

$$\frac{d\tau_i}{dt} = 0.15 \cdot (T - \tau_i) \qquad \text{with } i : 1, \ldots, 9 \tag{2}$$

When a new ingot with τ_{new} temperature enters into the furnace, the furnace temperature changes abruptly from T to $T - (T - \tau_{new})/ingots$, where $ingots$ is the number of ingots in the furnace.

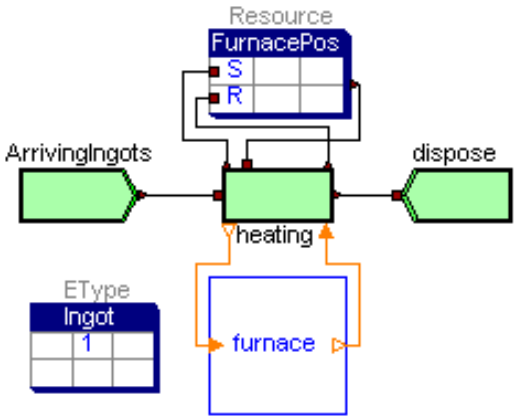


Fig. 6. Soaking-pit furnace system modeled using ARENALib.

The flowchart diagram of the system is shown in Fig. 6. Ingots arrive at the Create module, seize an available slot, are heated and finally leave the system. The heating process is represented using an ExternalProcess module from ARENALib. This discrete-event module represents the operation for seizing a free slot in the furnace and releasing it when finished. The diagram is constructed by drag-and-dropping the required elements from the ARENALib BasicProcess package into the model.

The furnace is described by Eqs. (1) and (2), and the abrupt changes in the furnace temperature triggered when new ingots enter the furnace. A detail of the Modelica code of the furnace model is shown in Listing 3. The reception of new ingots, the re-initialization of slot and furnace temperatures, and the management of the heating for each slot are shown.

```
1  algorithm
2    when IN <> pre(IN) then // new ingot received
3      while Posfree[i] > 0 loop // find free slot
4        i := mod(i,numFurnacePos)+1;
5      end while;
6      Posfree[i] := 1;// assign free slot
7      ingot[i] := IN;// record value of input ingot
8      // used to update furnace temp
9      newingot := newingot+1;
10   end when;
11   for i in 1:numFurnacePos loop //finished ingots?
12     when itemp[i] >= 2200 then
13       Posfree[i] := 0;
14       OUT := ingot[i];
15     end when;
16   end for;
17
18 equation
19   ftrate = der(ftemp);
20   ftrate = 2 * (2600 - ftemp); //furnace temp
21   for i in 1:numFurnacePos loop // temp of slots
22     itrate[i] = der(itemp[i]);
23     itrate[i] = if Posfree[i] == 0 then 0
24                 else 0.15 *(ftemp - itemp[i]);
25   end for;
26   // update furnace temp
27   when newingot <> pre(newingot) then
28     reinit(itemp[i],u);
29     reinit(ftemp,(ftemp-(ftemp-u)/sum(Posfree)));
30   end when;
```

Listing 3. Detail of Modelica code of furnace model.

The system has been simulated during 100 hours using Dymola 6.1. The time evolution of the furnace temperature (above) and the ingot temperatures (below) are shown in Fig. 7. The temperature of the furnace increases up to its maximum temperature (2600 ^{0}F), and decreases abruptly when a new ingot is inserted in a free slot. The temperatures of the ingots also increase up to the desired temperature (2200 ^{0}F). After that, the ingot leaves the slot but the temperature shown remains constant until a new ingot is inserted in the free slot. Slots are usually occupied sequentially, as it can be observed in the figure.

7 Emergency Hospital

This hybrid model is composed of two parts (see Fig. 8): the malaria spread model and the emergency hospital model. The former allows to estimate the malaria infection rate. The later is used to evaluate the use of the resources required to treat the infected people. In this case study, the continuous-time model of the malaria spread serves as input for the discrete-time model of the emergency hospital. The connection is performed using the Create module from the ARENALib library. This combination can be also described using Arena/SIMAN (i.e., using the DETECT SIMAN block). However, the description of the continuous-time model using Arena/SIMAN requires to use a general-purpose programming language, such as Visual Basic, C or FORTRAN, while Modelica provides all the required functionality.

7.1 Malaria Spread Model

The malaria spread process is described as a continuous-time model composed of the following equations (Chitnis et al., 2006):

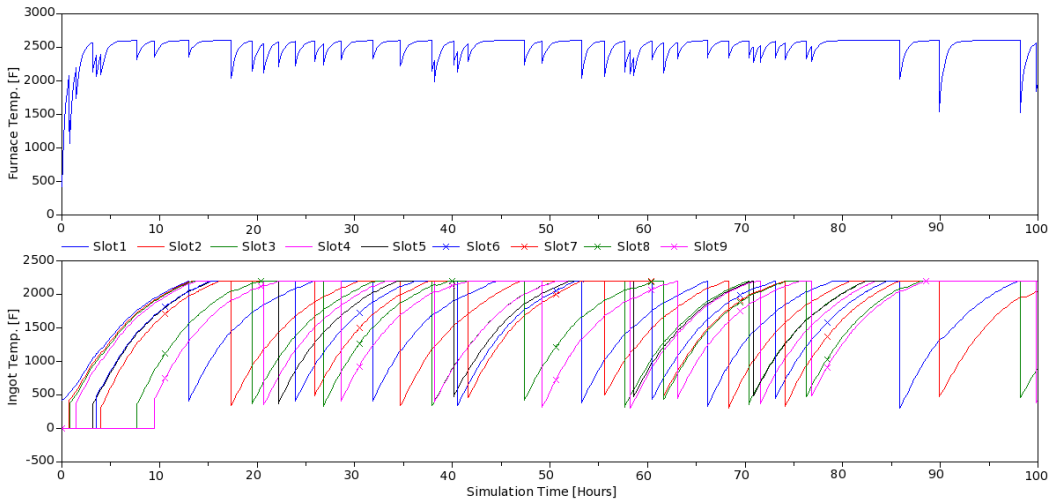


Fig. 7. Soaking-pit furnace system: furnace (above) and ingot (below) temperatures expressed in degrees Fahrenheit.

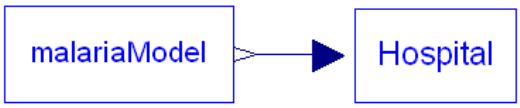


Fig. 8. Connection between the malaria spread model and the emergency hospital model.

$$\frac{dS_h}{dt} = \Lambda_h + \psi_h \cdot N_h + \rho_h \cdot R_h - \lambda_h(t) \cdot S_h - f_h(N_h) \cdot S_h \tag{3}$$
$$\frac{dE_h}{dt} = \lambda_h(t) \cdot S_h - \nu_h \cdot E_h - f_h(N_h) \cdot E_h, \tag{4}$$
$$\frac{dI_h}{dt} = \nu_h \cdot E_h - \gamma_h \cdot I_h - f_h(N_h) \cdot I_h - \delta_h \cdot I_h \tag{5}$$
$$\frac{dR_h}{dt} = \gamma_h \cdot I_h - \rho_h \cdot R_h - f_h(N_h) \cdot R_h \tag{6}$$
$$\frac{dS_v}{dt} = \psi_v \cdot N_v - \lambda_v(t) \cdot S_v - f_v(N_v) \cdot S_v \tag{7}$$
$$\tag{8}$$

$$\frac{dE_v}{dt} = \lambda_v(t) \cdot S_v - \nu_v \cdot E_v - f_v(N_v) \cdot E_v \tag{9}$$
$$\frac{dI_v}{dt} = \nu_v \cdot E_v - f_v(N_v) \cdot I_v \tag{10}$$

where $f_h(N_h) = \mu_{1h} + \mu_{2h} \cdot N_h$ is the per capita density-dependent death and emigration rate for humans and $f_v(N_v) = \mu_{1v} + \mu_{2v} \cdot N_v$ is the per capita density-dependent death rate for mosquitoes. The descriptions of the state variables and the parameters for the model are shown in Tables 1 and 2, respectively.

The total population rates are

$$N_h = S_h + E_h + I_h + R_h \tag{11}$$
$$N_v = S_v + E_v + I_v \tag{12}$$

The infection rates are

$$\lambda_h = b_h(N_h, N_v) \cdot \beta_{hv} \cdot \frac{I_v}{N_v} \tag{13}$$
$$\lambda_v = b_v(N_h, N_v) \cdot (\beta_{vh} + \tilde{\beta}_{vh} \cdot \frac{R_h}{N_h}) \tag{14}$$

where λ_h is the force of infection from mosquitoes to humans and λ_v is the force of infection from humans to mosquitoes. In these equations, $b_h = b(N_h, N_v)/N_h$

Name	Description
S_h:	Number of susceptible humans.
E_h:	Number of exposed humans.
I_h:	Number of infectious humans.
R_h:	Number of recovered (immune and asymptomatic, but slightly infectious) humans.
S_v:	Number of susceptible mosquitoes.
E_v:	Number of exposed mosquitoes.
I_v:	Number of infectious mosquitoes.
N_h:	Total human population.
N_v:	Total mosquito population.

Table 1
State variables of the malaria spread model (Chitnis et al., 2006).

describes the number of mosquito bites that one human can have per unit time, and $b_v = b(N_h, N_v)/N_v$ describes the number of human bites one mosquito can have per unit time, with $b(N_h, N_v)$ that defines the total number of mosquito bites on humans, as:

$$b = b(N_h, N_v) = \frac{\sigma_v \cdot \sigma_h}{\sigma_v(N_v/N_h) + \sigma_h} \cdot N_v \tag{15}$$

Eqs. (3)–(15) can be directly translated into the Modelica code shown in Listing 4. Note that the translation of these equations to Modelica, given that the Modelica operator `der()` describes the time derivative of a variable, is straight forward.

```
der(Sh) = Ah+Psih*Nh+Rhoh*Rh-Lambdah*Sh-Fh*Sh;
der(Eh) = Lambdah*Sh - Vh*Eh - Fh*Eh;
der(Ih) = Vh*Eh - Gammah*Ih - Fh*Ih - Deltah*Ih;
der(Rh) = Gammah*Ih - Rhoh*Rh - Fh*Rh;
der(Sv) = Psiv*Nv - Lambdav*Sv - Fv*Sv;
der(Ev) = Lambdav*Sv - Vv*Ev - Fv*Ev;
der(Iv) = Vv*Ev - Fv*Iv;
Fh = Mu1h + Mu2h*Nh;
Fv = Mu1v + Mu2v*Nv;
Nh = Sh + Eh + Ih + Rh;
Nv = Sv + Ev + Iv;
Lambdah = bh*Betahv*(Iv/Nv);
Lambdav = bv*(Betavh*(Ih/Nh)+Betatildevh*(Rh/Nh));
bh = b/Nh;
bv = b/Nv;
b = (Sigmav*Sigmah/(Sigmav*(Nv/Nh) + Sigmah))*Nv;
```

Listing 4. Modelica code for malaria spread model.

Name	Description
Λ_h:	Immigration rate of humans. ($Humans \times Time^{-1}$)
ψ_h:	Per capita birth rate of humans. ($Time^{-1}$)
ψ_v:	Per capita birth rate of mosquitoes. ($Time^{-1}$)
σ_v:	Number of times one mosquito would want to bite humans per unit time, if humans were freely available. This is a function of the mosquito's gonotrophic cycle (the amount of time a mosquito requires to produce eggs) and its anthropophilic rate (its preference for human blood). ($Time^{-1}$)
σ_h:	The maximum number of mosquito bites a human can have per unit time. This is a function of the human's exposed surface area. ($Time^{-1}$)
β_{hv}:	Probability of transmission of infection from an infectious mosquito to a susceptible human, given that a contact between the two occurs. ($Unitless$)
β_{vh}:	Probability of transmission of infection from an infectious human to a susceptible mosquito, given that a contact between the two occurs. ($Unitless$)
$\tilde{\beta}_{vh}$:	Probability of transmission of infection from a recovered (asymptomatic carrier) human to a susceptible mosquito, given that a contact between the two occurs. ($Unitless$)
ν_h:	Per capita rate of progression of humans from the exposed state to the infectious state. $1/\nu_h$ is the average duration of the latent period. ($Time^{-1}$)
ν_v:	Per capita rate of progression of mosquitoes from the exposed state to the infectious state. $1/\nu_v$ is the average duration of the latent period. ($Time^{-1}$)
γ_h:	Per capita recovery rate for humans from the infectious state to the recovered state. $1/\gamma_h$ is the average duration of the infectious period. ($Time^{-1}$)
δ_h:	Per capita disease-induced death rate for humans. ($Time^{-1}$)
ρ_h:	Per capita rate of loss of immunity for humans. $1/\rho_h$ is the average duration of the immune period. ($Time^{-1}$)
μ_{1h}:	Density-independent part of the death (and emigration) rate for humans. ($Time^{-1}$)
μ_{2h}:	Density-dependent part of the death (and emigration) rate for humans. ($Humans^{-1} \times Time^{-1}$)
μ_{1v}:	Density-independent part of the death rate for mosquitoes. ($Time^{-1}$)
μ_{2v}:	Density-dependent part of the death rate for mosquitoes. ($Mosquitoes^{-1} \times Time^{-1}$)

Table 2
Parameters of the malaria model and their units (Chitnis et al., 2006).

7.2 Hospital Model

The emergency hospital has been modeled using ARENALib. The model is based in the hospital model included as an example in the Arena simulation environment. The flowchart diagram of the model, shown in Fig. 9a, is described below.

The number of infected people calculated using the malaria spread model corresponds to the value of the port named *Infected*. The value of that variable is quantized using the quantizerUP model from the DEVSLib library, which generates a message every time the Infected variable increases in one unit (i.e., a new person is infected). A create block, from the SIMANLib library, is used to generate an entity that corresponds to the newly infected person.

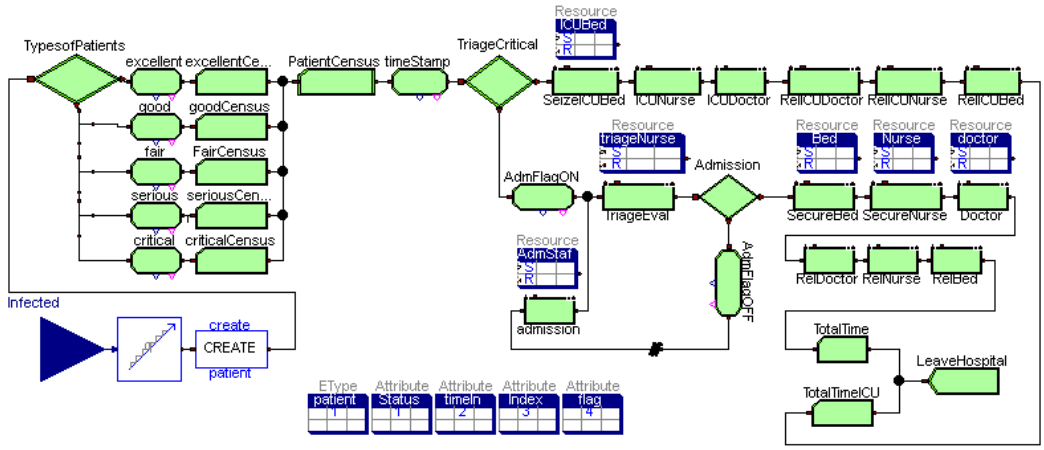
After that, patients are randomly distributed between different types: excellent (20%), good (35%), fair (30%), serious (10%) and critical (5%). Critical patients include those with severe malaria symptoms and those who need parenteral administration of anti-malarian drugs (WHO, 2010). These patients are directly transferred to the ICU area, where dedicated beds are available. The treatment of critical patients requires a dedicated nurse and supervision by a doctor, and the duration changes from one to three days. Other patients have to go through the admission procedure before being treated. Once a bed is assigned to the patient, he is attended by a doctor and after that a nurse teaches him how to self-administrate the drugs. When the treatment is finished, the bed is released and the patient leaves the hospital. This model automatically records statistics for the type of patients that arrive, the utilization of resources and the time spent by the patients in the hospital.

7.3 Validation and Simulation

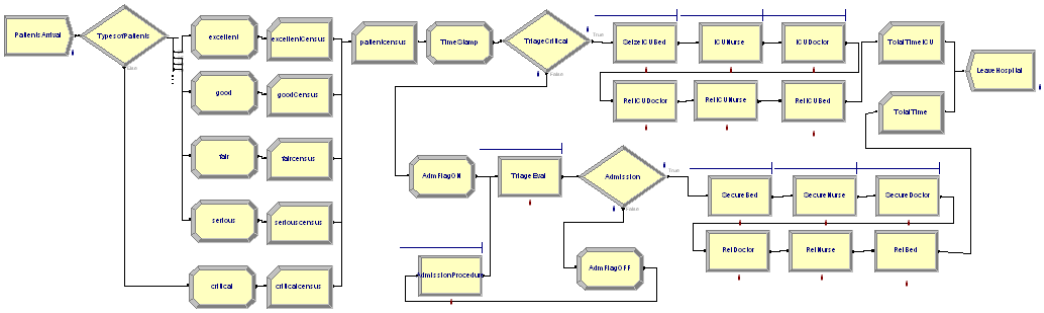
The Modelica model of the malaria spread has been compared with the model developed in Chitnis et al. (2006). The simulation results of both models are equivalent, using the initial values and parameters shown in Tables 3 and 4, and are shown in Fig. 10.

$S_h = 400$ humans	$S_v = 1000$ mosquitoes
$E_h = 10$ humans	$E_v = 100$ mosquitoes
$I_h = 30$ humans	$I_v = 50$ mosquitoes
$R_h = 0$ humans	

Table 3
Initial values for the malaria spread model (Chitnis et al., 2006).



a)



b)

Fig. 9. Emergency department of a hospital modeled using: a) ARENALib; and b) Arena.

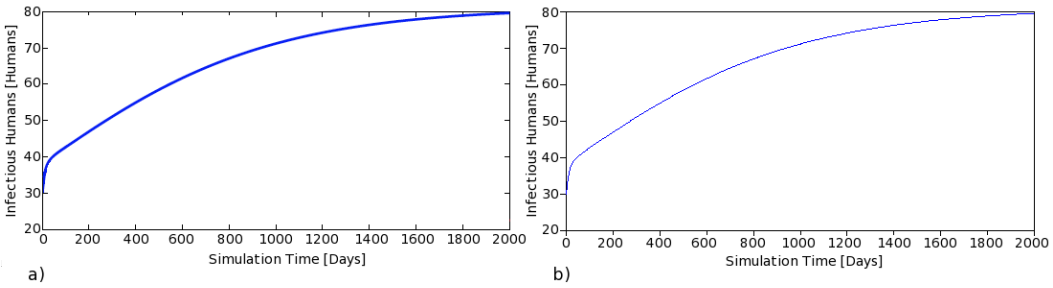


Fig. 10. Simulation results for malaria spread model using the initial values and parameters in Tables 3 and 4: a) model from Chitnis et al. (2006); and b) Modelica model.

The validation of the emergency hospital model has been performed by comparing the developed ARENALib model with an equivalent model developed using Arena, shown in Fig. 9b. A Create module has been used in both models as a source of patients (substituting the *Infected* port, the quantizerUP model and the SIMANLib create block, shown in Fig. 9a), since this comparison does not involve the developed malaria spread model. Both models have been simulated during 8000 time units to observe the system in steady-state. Some

Name and Value		Unit
$\Lambda_h = 3.285 \times 10^{-2}$		humans $\times$ time^{-1}
$\psi_h = 7.666 \times 10^{-5}$	$\psi_v = 0.4000$	time^{-1}
$\beta_{vh} = 0.8333$	$\beta_{hv} = 2.000 \times 10^{-2}$	unitless
$\tilde{\beta}_{vh} = 8.333 \times 10^{-2}$		unitless
$\sigma_h = 0.6000 \times 10^{-2}$	$\sigma_v = 18.00$	time^{-1}
$\nu_h = 8.333 \times 10^{-2}$	$\nu_v = 0.1000$	time^{-1}
$\gamma_h = 3.704 \times 10^{-3}$		time^{-1}
$\delta_h = 3.454 \times 10^{-4}$		time^{-1}
$\rho_h = 1.460 \times 10^{-2}$		time^{-1}
$\mu_{1h} = 4.212 \times 10^{-5}$	$\mu_{1v} = 0.1429$	time^{-1}
$\mu_{2h} = 1.000 \times 10^{-7}$		humans^{-1} $\times$ time^{-1}
$\mu_{2v} = 2.279 \times 10^{-4}$		mosquitoes^{-1} $\times$ time^{-1}

Table 4
Parameters of the malaria spread model (Chitnis et al., 2006).

statistical indicators, regarding the total time spent in the system, the waiting time in the queues and the utilization of resources, calculated in both models are shown in Table 5. The simulation results obtained from the ARENALib model are always in the range of the half-width interval calculated by Arena.

Indicator	Arena	Half-width	ARENALib	Unit
Patient Total Time	0.2534	0.0034	0.2501	time
Patient Total ICU time	2.1514	0.0627	2.1884	time
Seize ICU bed.WaitingTime	0.1057	0.0381	0.1276	time
TriageEval.WaitingTime	0.0058	0.0005	0.0053	time
Nurse.NumberBusy	0.2214	0.0095	0.2138	%
Doctor.NumberBusy	0.0397	0.0014	0.0396	%
AdmStaff.NumberBusy	0.0189	0.0004	0.0189	%
TriageNurse.NumberBusy	0.0872	0.0025	0.0856	%
Bed.NumberBusy	0.1253	0.0044	0.1225	%
ICUBed.NumberBusy	0.1066	0.0098	0.1012	%

Table 5
Simulation results from the comparison of the emergency hospital models.

Finally, the complete system, combining the malaria spread model and the

$S_h = 40000$ humans	$S_v = 100000$ mosquitoes
$E_h = 1000$ humans	$E_v = 10000$ mosquitoes
$I_h = 0$ humans	$I_v = 5000$ mosquitoes
$R_h = 0$ humans	

Table 6
Initial values for the epidemic simulation in the malaria spread model.

emergency hospital, has been simulated during 100 time units. In order to simulate an epidemic situation in the malaria model, the initial values shown in Table 6 have been used. In this situation the number of infected humans increases rapidly, thus creating a saturation in the hospital. This saturation can be observed in the number of patients waiting for a bed, specially ICU beds, whose evolution is shown in Fig. 11.

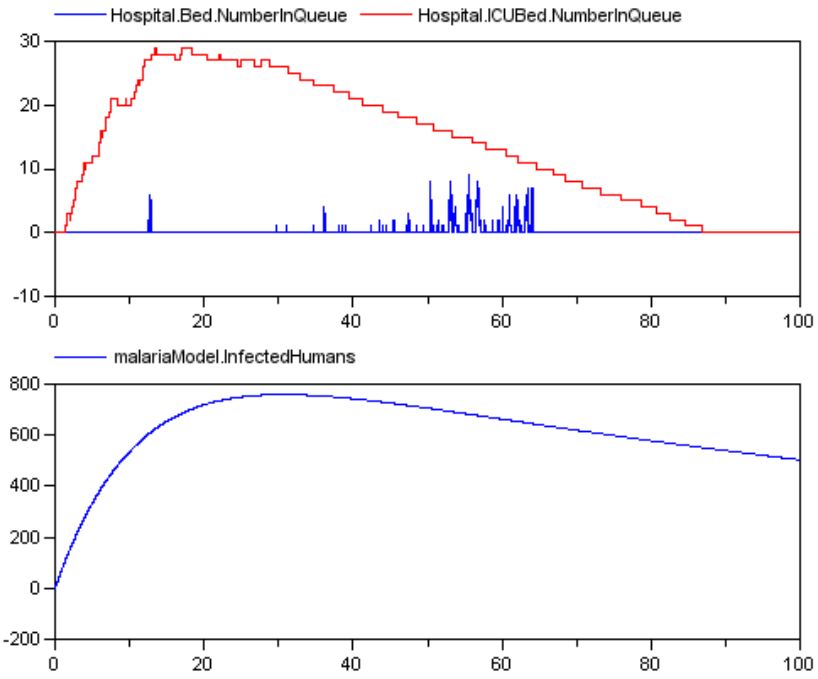


Fig. 11. Evolution of the number of infected people and the number of patients waiting for a bed.

8 Conclusions

The SIMANLib and ARENALib Modelica libraries facilitate the application of the Arena modeling methodology in Modelica. These libraries reproduce some modeling functionality of SIMAN and Arena, providing Modelica components to describe the flowchart diagram and the static information of the model. ARENALib reproduces most of the modeling functionality of the *Ba-*

sic Process panel of Arena. The SIMAN blocks supported by SIMANLib include: Create, Dispose, Queue, Seize, Delay, Release, Branch, Count, Assign and Tally. Some SIMAN elements supported by SIMANLib are: EntityType, Queue, Resource, Attribute, Variable, Counter, DStat and Tally. In addition, ARENALib and SIMANLib provide components describing the connection between the discrete-event models composed using these libraries, and models composed using other Modelica models. This feature facilitates combining ARENALib/SIMANLib models with models developed applying other modeling methodologies supported by Modelica (e.g., DAE-hybrid models).

The following tasks have been completed for developing SIMANLib and ARENALib:

(1) The behavior of the supported modules, blocks and elements has been described using Parallel DEVS. In particular, the SIMANLib blocks have been described as atomic Parallel DEVS models and the ARENALib modules as coupled Parallel DEVS models.
(2) The functionality of the components required to communicate ARENALib/SIMANLib models with other Modelica models has been specified.
(3) The libraries have been programmed in Modelica. The translation of the model formal description into Modelica code has been facilitated by the use of the DEVSLib Modelica library. The description of an atomic Parallel DEVS model using DEVSLib is very close to its formal specification i.e., it is performed by describing each element of the tuple. The transition, output and time-advance functions are specified using Modelica functions. This facilitates the model description and the understanding of the developed models. The description of coupled Parallel DEVS models with DEVSLib also matches completely with its formal specification. It is performed simply by connecting the corresponding ports of the component Parallel DEVS models.

    The following additional functionality was required to support the Arena modeling methodology in Modelica: a) management of the information that describes the entities in the system; and b) description of an information storage structure to facilitate the management of variable-size data generated during the simulation run (i.e., statistical indicators, global variables and user-defined attributes for the entities). Two external libraries written in C code have been programmed to support these requirements.
(4) The libraries have been validated by comparing the simulation results of the developed models with the ones obtained using equivalent models developed using Arena/SIMAN. SIMANLib and ARENALib include a package named "BookExamples" that includes the implementation of some models described in Pegden et al. (1995) and Kelton et al. (2007), respectively.

The implementation and design of SIMANLib and ARENALib have been discussed in this manuscript. Two case studies have been presented to illustrate the library use for describing hybrid models. The soaking-pit furnace system illustrates the use of the ExternalProcess module included in ARENALib. This module can be used to describe the integration of external processes, described using other Modelica libraries, with ARENALib models. An emergency hospital is modeled to present the possible interactions between continuous-time and ARENALib/SIMANLib models, where the generation of entities depends on the evolution of the continuous-time model. The simulation of these models has been compared with equivalent existing models, or models constructed using SIMAN and Arena. The results obtained are equivalent.

References

Åström, K. J., Elmqvist, H., Mattsson, S. E., 1998. Evolution of continuous-time modeling and simulation. In: Proceedings of the 12^{th} European Simulation Multiconference (ESM'98). Manchester, UK, pp. 9–18.

Cellier, F. E., Kofman, E., 2006. Continuous System Simulation. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Chitnis, N., Cushing, J. M., Hyman, J. M., 2006. Bifurcation analysis of a mathematical model for malaria transmission. SIAM Journal of Applied Mathematics 67, 24–45.

Dynasim AB, 2006. Dymola, Dynamic Modeling Laboratory. User's manual. URL `http://www.dymola.com/`

Fabricius, S. M., Badreddin, E., 2002. Hybrid dynamic plant performance analysis supported by extensions to the Petri Net library in Modelica. In: Proceedings of the 4^{th} Asian control Conference (ASCC). Singapore, pp. 41–50.

Ferreira, J., de Oliveira, J. E., 1999. Modelling hybrid systems using StateCharts and Modelica. In: Proceedings of the 7^{th} IEEE International Conference on Emerging Technologies and Factory Automation. pp. 1063–1069.

Forrester, J. W., 1969. Principles of Systems. Waltham, MA, USA.

Kelton, W. D., Sadowski, R. P., Sturrock, D. T., 2007. Simulation with Arena, 4^{th} Edition. McGraw-Hill, Inc., New York, NY, USA.

Law, A. M., 2007. Simulation Modelling and Analysis, 4^{th} Edition. McGraw-Hill, New York, NY, USA.

Mikler, J., Engelson, V., 2003. Simulation for operation management: Object oriented approach using Modelica. In: Proceedings of the 3^{rd} International Modelica Conference. Linköping, Sweden, pp. 207–214.

Modelica Association, 2012. Modelica - An unified object-oriented language for physical systems modeling. Language specification version 3.1. URL `http://www.modelica.org/documents`

Modelica Libraries, 2012. Modelica free and comercial libraries.

URL http://www.modelica.org/libraries

Mosterman, P. J., Otter, M., Elmqvist, H., 1998. Modelling Petri Nets as local constraint equations for hybrid systems using Modelica. In: Proceedings of the Summer Computer Simulation Conference. pp. 314–319.

Otter, M., Årzén, K.-E., Dressler, I., 2005. StateGraph - a Modelica library for hierarchical state machines. In: Proceedings of the 4^{th} International Modelica Conference. Hamburg, Germany, pp. 569–578.

Pegden, C. D., Sadowski, R. P., Shannon, R. E., 1995. Introduction to Simulation Using SIMAN. McGraw-Hill, Inc., New York, NY, USA.

Pulecchi, T., Casella, F., 2008. HyAuLib: modelling hybrid automata in Modelica. In: Proceedings of the 6^{th} International Modelica Conference. Bielefeld, Germany, pp. 239–246.

Robinson, S., Nance, R. E., Paul, R. J., Pidd, M., Taylor, S. J., 2004. Simulation model reuse: definitions, benefits and obstacles. Simulation Modelling Practice and Theory 12 (7-8), 479 – 494, simulation in Operational Research.

Sanz, V., 2010. Hybrid system modeling using the Parallel DEVS formalism and the Modelica language. Ph.D. thesis, E.T.S.I. Informtica, UNED, Madrid, Spain.

Sanz, V., Urquia, A., Cellier, F. E., Dormido, S., 2010. System modeling using the Parallel DEVS formalism and the Modelica language. Simulation Modeling Practice and Theory 18 (7), 998–1018.

Wetter, M., 2011. Co-simulation of building energy and control systems with the building controls virtual test bed. Journal of Building Performance Simulation 4, 185–203.

WHO, 2010. World health organization guidelines for the treatment of malaria.

Zeigler, B. P., Kim, T. G., Prähofer, H., 2000. Theory of Modeling and Simulation. Academic Press, Inc., Orlando, FL, USA.