Joint publication on adapting industrial simulations to obtain virtual experiments for learning

Denis Gillet, Nalin Navarathna, Carla Martin-Villalba, Alfonso Urquia, Sebastián Dormido

▶ **To cite this version:**

Denis Gillet, Nalin Navarathna, Carla Martin-Villalba, Alfonso Urquia, Sebastián Dormido. Joint publication on adapting industrial simulations to obtain virtual experiments for learning. 2007. hal-00591558

HAL Id: hal-00591558
https://hal.science/hal-00591558

Submitted on 10 May 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Network of Excellence Professional Learning

PROLEARN

European Sixth Framework Project

D.10.11
Joint publication on adapting industrial simulations to obtain virtual experiments for learning

Work Package	*WP10*
Document Number	*10.11.2*
Author(s):	*Associated partners from UNED*
Editor (Internal reviewer)	*Denis GILLET*
Status	*Final*
Date of draft	*Month 48 (Dec 2007)*

Summary

The development of online experiments and especially virtual laboratories (i.e. interactive simulation of real equipments) is a complex and resource-consuming process. Their availability is however a prerequisite to any serious learning activities in sciences and engineering.

The trend for reducing the implementation overhead for educators is to rely on professional simulation packages on top of which technology-enhanced learning layers are added. The first type of professional simulation packages used in education includes game development engines that provide suitable functionalities for effectively defining scenarios and rendering complex entities. The difficulty in this case is to develop the dynamical model of the scientific or technical objects to be simulated. The second type of professional simulation packages includes professional model libraries targeting at specific application domains. The difficulty here is related to the development of the interactive user interfaces (views).

In the framework of the PROLEARN network of excellence, core and associated partners have developed and shared best practices for the development and the implementation of virtual laboratories relying on the EasyJava Simulations framework (http://www.um.es/fem/Ejs/), which is part of the Open Source Physics project. The EasyJava Simulations framework especially facilitates the design of virtual laboratories with highly interactive user interfaces (views).

The present document presents a recent extension of this work that takes advantage of the freely available, object-oriented modelling language Modelica that strongly facilitates the reuse of industrial models in educational virtual laboratories, while keeping the benefits of the EasyJava Simulations. It corresponds to a state-of-the-art approach shared by PROLEARN partners.

This document is the preprint of an article accepted for publication in the Mathematical and Computer Modelling of Dynamical Systems Journal (copyright Taylor & Francis) that is available online at http://journalsonline.tandf.co.uk/.

The PROLEARN Consortium

1. Universität Hannover, Learning Lab Lower Saxony (L3S), Germany
2. Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI), Germany
3. Open University (OU), UK
4. Katholieke Universiteit Leuven (K.U.Leuven) / ARIADNE Foundation, Belgium
5. Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V. (FHG), Germany
6. Wirtschaftsuniversität Wien (WUW), Austria
7. Universität für Bodenkultur, Zentrum für Soziale Innovation (CSI), Austria
8. École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
9. Eidgenössische Technische Hochschule Zürich (ETHZ), Switzerland
10. Politecnico di Milano (POLIMI), Italy
11. Jožef Stefan Institute (JSI), Slovenia
12. Universidad Polictécnica de Madrid (UPM), Spain
13. Kungl. Tekniska Högskolan (KTH), Sweden
14. National Centre for Scientific Research “Demokritos” (NCSR), Greece
15. Institut National des Télécommunications (INT), France
16. Hautes Etudes Commerciales (HEC), France
17. Technische Universiteit Eindhoven (TU/e), Netherlands
18. Rheinisch-Westfälische Technische Hochschule Aachen (RWTH), Germany
19. Helsinki University of Technology (HUT), Finland
20. information – multimedia – communication AG (imc), Germany
21. Open University of the Netherlands (OUNL), Netherlands
22. University of Warwick (UW), UK

Document Control

Title: Joint publication on adapting industrial simulations to obtain virtual experiments for learning

Editor: Denis GILLET

E-mail: denis.gillet@epfl.ch

AMENDMENT HISTORY

Version	Date	Author	Description/Comments
1.0	24.11.2007	Nalin Navarathna	Initial version proposed by KTH
2.0	12.12.2007	Carla Martin-Villalba Alfonso Urquia Sebastian Dormido	New version from UNED
2.1	12.01.2008	Denis Gillet	Minor changes in the Summary
2.2	13.01.2008	Denis Gillet	Comment from KTH taken into account

Legal Notices

The information in this document is subject to change without notice.

The Members of the PROLEARN Consortium make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the PROLEARN Consortium shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

An approach to virtual-lab implementation using Modelica

CARLA MARTIN-VILLALBA*, ALFONSO URQUIA and SEBASTIAN DORMIDO

Departamento de Informática y Automática, UNED
Juan del Rosal 16, 28040 Madrid, Spain

A novel approach to the implementation of interactive virtual-labs is proposed. The virtual-lab is completely described in Modelica language and translated using Dymola. To achieve this goal, a systematic methodology to transform any Modelica model into a formulation suitable for interactive simulation has been developed. In addition, *VirtualLabBuilder* Modelica library has been programmed. This library contains a set of Modelica models of visual interactive elements (i.e., containers, animated geometric shapes and interactive controls) that allows easy creation of the virtual-lab view (i.e., the model-to-user interface).

This approach has two strong points. Firstly, it allows taking advantage of the Modelica capabilities for multi-domain modelling and model reuse. In particular, existing Modelica libraries for modelling of physical systems can be reused in order to build the virtual-lab models. Secondly, *VirtualLabBuilder* library allows describing the virtual-lab view with Modelica, which facilitates its development, maintenance and reuse. The proposed approach is discussed in this manuscript and it is illustrated by means of the following case study: a virtual-lab describing the thermodynamic behaviour of a solar house.

VirtualLabBuilder library can be freely downloaded from http://www.euclides.dia.uned.es/

Keywords: Modelica; Virtual laboratory; Control education; Interactive simulation; Simulation animation

AMS Subject Classification: 97U04; 93A04; 93C04

1. Introduction

A virtual-lab is a distributed environment of simulation and animation tools, intended to perform the interactive simulation of a mathematical model. Virtual-labs supporting interactive simulations are effective educational resources [1]. During the interactive simulation run, students can change the value of some selected inputs, parameters and state variables of the model, perceiving instantly how these changes affect the model dynamic. An arbitrary number of actions can be made on the model during a given simulation run. Virtual-labs allow students to play an active role in their learning process and this promotes their motivation to study the subject.

Virtual-labs are composed of *introduction*, *model* and *view*. The *introduction* is the documentation (frequently a set of HTML pages) discussing the concepts and phenomena illustrated by the virtual-lab, how to experiment with the lab, and also proposing some activities or exercises [2]. The *model* is the mathematical model representing the relevant behaviour of the system under study. The *view* is the user-to-model interface. It is intended to provide a visual representation of the simulated model behaviour and to facilitate the user's interactive actions on the model during the simulation run. The graphical properties of the view elements are linked to the model variables, producing a bi-directional flow of information between the view and the model. Any change of a model variable value is automatically displayed by the view. Reciprocally, any user interaction with the view automatically modifies the value of the corresponding model variable.

There are many software tools specifically intended to facilitate the implementation of virtual-labs. Two of them are Easy Java Simulations [3, 4] and Sysquake [5]. The strong point of these tools is that they allow easy creation of the virtual-lab view. Their weak point is the model definition, that requires explicit state models (ODE). This restriction strongly conditions the modelling task, which requires a considerable effort from the virtual-lab developer.

Modelica [6] is a freely available, object-oriented modelling language that supports the physical modelling paradigm [7]. Models are mathematically described by differential and algebraic equations (DAE), and

*Corresponding author. Email: carla@dia.uned.es

discrete equations. Modelica supports a declarative (i.e., non-causal) description of the model. Therefore, the use of Modelica reduces considerably the modelling effort and permits better reuse of the models. In addition, a number of free and commercial Modelica libraries in different domains are available [6], including electrical, mechanical, thermo-fluid and physical-chemical.

However, neither Modelica language nor Modelica simulation environments (i.e., Dymola [8], OpenModelica [9], etc.) support interactive simulation. As a consequence, extending Modelica capabilities in order to facilitate interactive simulation (i.e., virtual-lab implementation) is an open research field [10–13]. Previous work on this topic addresses the combined use of Modelica/Dymola and other software tools [11–13]: the virtual-lab model is described using Modelica and the virtual-lab view is implemented using software tools suited for building interactive user interfaces. In particular, the combined use of Modelica/Dymola, Matlab and Easy Java Simulations is proposed in [11–13]; and the combined use of Modelica/Dymola and Sysquake is proposed in [12].

The goal of the work discussed in this manuscript is to facilitate the description of virtual-labs using Modelica language. To achieve this goal, the following two tasks have been completed [14]:

(i) A systematic methodology to transform any Modelica model into a formulation suitable for interactive simulation has been proposed. Modelica models adapted according to this methodology can be used to set up interactive virtual-labs.
(ii) *VirtualLabBuilder* Modelica library has been designed and programmed. It includes Modelica models implementing graphic interactive elements, such as containers, animated geometric shapes and interactive controls. These models allow the virtual-lab developer: (1) to compose the view; and (2) to link the visual properties of the virtual-lab view with the model variables. The interactive graphic interface is automatically generated during the model initialization process. The components of the library contain the code required to perform the bidirectional communication between the view and the model. In addition, *VirtualLabBuilder* library supports including in the virtual-lab an introductory part, which is composed of HTML pages.

An overview of the proposed approach to virtual-lab implementation using Modelica is provided in section 2. The methodology to transform any Modelica model into a formulation suitable for interactive simulation is discussed in section 3. The architecture and use of *VirtualLabBuilder* library is described in section 4. Finally, the application of the proposed approach is illustrated by means of the following case study: the implementation of a virtual-lab describing the thermodynamic behaviour of a solar house (section 5). Other case studies can be found in [14, 15].

2. Overview of the proposed approach

The virtual-lab definition includes the description of the introduction, the model, the view, and the bidirectional flow of information between the model and the view. Next, the virtual-lab definition process is outlined.

(i) **Virtual-lab model.** Any Modelica model can be transformed into another Modelica model suitable for interactive simulation. A systematic methodology to perform this transformation is proposed in section 3. Essentially, it consists in modifying the model so that all the variables that need to be changed interactively during the simulation (i.e., the *interactive variables*) are formulated as state variables. In particular, parameters are redefined as time-dependent variables whose time-derivative is equal to zero. Input variables are reformulated analogously in order to become interactive variables. Modelica's *when* clause and *reinit* operator allow describing instantaneous changes in the value of the state variables. This feature is exploited in order to perform the instantaneous changes in the value of the interactive variables produced by the user's interaction. Some of these model manipulations could be performed automatically by a software tool. However, at the present time, they have to be carried out manually by the virtual-lab developer.
(ii) **Virtual-lab view.** The virtual-lab developer has to define a Modelica class describing the virtual-lab view. This class has to extend another class, named PartialView, that is included in *VirtualLabBuilder*

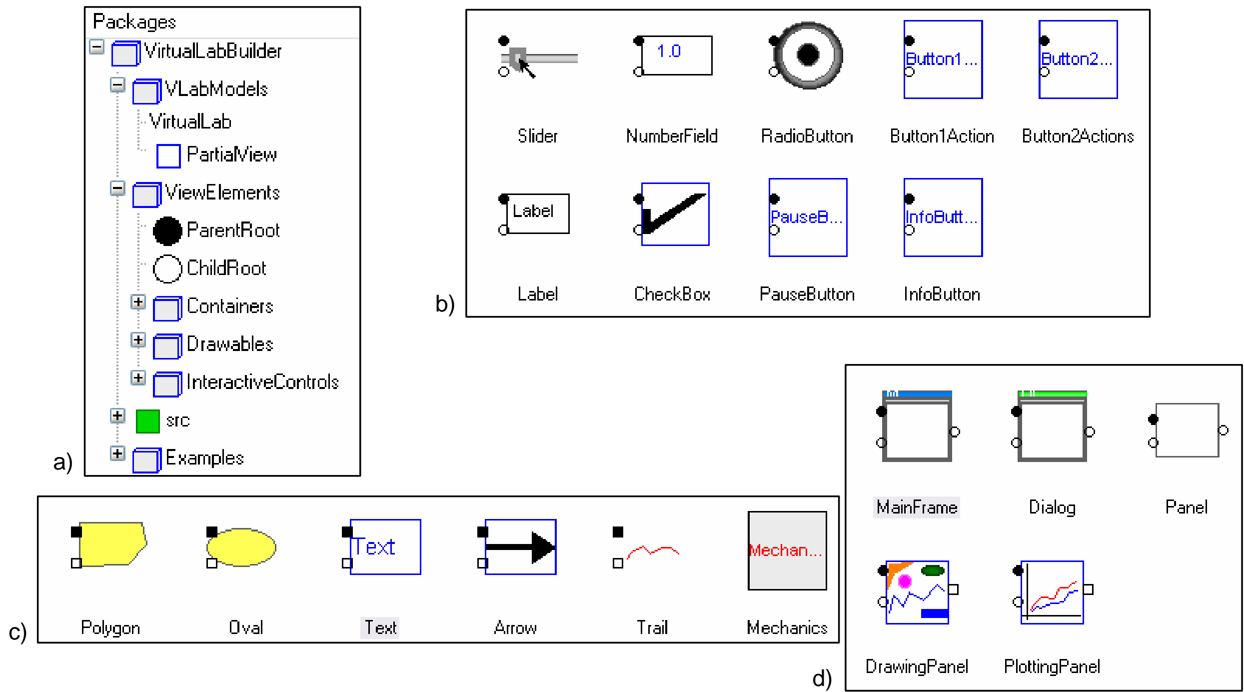

Figure 1. *VirtualLabBuilder* library: a) General structure; and Classes within the following packages: b) InteractiveControls; c) Drawables; and d) Containers.

library (see figure 1a). PartialView class contains a pre-defined component: the *root element* for the view description. The classes describing the graphic components are within the InteractiveControls, Drawables and Containers packages of *VirtualLabBuilder* library (see figures 1b, 1c and 1d respectively). The virtual-lab designer has to compose the virtual-lab view class by instantiating and connecting the required graphic components. The graphic components have to be connected forming a structure, whose root is the *root element*. The connections among the graphic components determine their layout in the virtual-lab view. *VirtualLabBuilder*'s graphic components and their connection rules are discussed in section 4.

(iii) **Virtual-lab set up.** The virtual-lab developer has to define a Modelica class describing the complete virtual-lab. This class has to extend the VirtualLab class, which is within the *VirtualLabBuilder* library (see figure 1a). VirtualLab class has two parameters: the class describing the virtual-lab model, and the class describing the virtual-lab view. These two classes have been programmed in Steps (i) and (ii) respectively. The virtual-lab designer has to set the value of these parameters by writing the name of these two classes. In addition, he has to specify how the variables of the model and the view Modelica classes are linked. This is accomplished by writing the required Modelica equations inside the Modelica class defining the complete virtual-lab.

(iv) **Virtual-lab translation and execution.** The virtual-lab developer needs to translate using Dymola [8] an instance of the Modelica class defined in Step (iii) into an executable file (i.e., *dymosim.exe* file). The virtual-lab is started by executing this file.

(v) **Automatic code generation and run.** At the beginning of the simulation run, some calculations are performed in order to solve the model at the initial time. The *initial sections* of the Modelica model describing the virtual-lab are evaluated. In particular, the *initial sections* of the interactive graphic objects composing the virtual-lab view class and of the PartialView class are executed. These *initial sections* contain calls to Modelica functions, which encapsulate calls to external C-functions. These C-functions are Java-code generators. As a result, during the model initialization, the Java code of the virtual-lab view is automatically generated, compiled, packed into a jar file and executed. Also, the communication procedure between the model and the view is automatically set up. This communication is based on a client-server architecture: the C-program generated by Dymola [8] (i.e., *dymosim.exe*, see

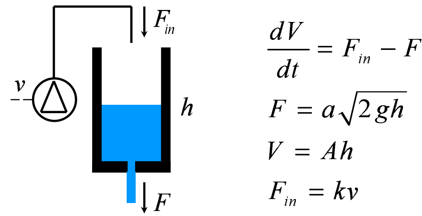

Figure 2. Model used to illustrate the translation from the *physical model* to the *interactive model*.

Step (iv)) is the server and the Java program (which has been automatically generated during the model initialization) is the client. Once the jar file is executed, the initial layout of the virtual-lab view is displayed and the client-server communication is established. Then, the model simulation starts. During the simulation run, there is a bi-directional flow of information between the model and the view.

3. Model description for interactive simulation using Modelica language

A methodology for transforming any Modelica model into a description suitable for interactive simulation is proposed in this section. The following terminology will be used. The original model of the system is called *physical model*, and its reformulation for interactive simulation is called *interactive model*.

The model shown in figure 2 will be used to illustrate the discussion. The voltage applied to the pump (v) is an *input variable* (i.e., its value is not calculated from the model equations). The cross-sections of the tank (A) and the outlet hole (a), the pump parameter (k) and the gravitational acceleration (g) are *parameters* (i.e., time-independent quantities of the model). The liquid volume (V), the input and output flows (F_{in}, F), and the liquid level (h) are *time-dependent variables* of the physical model.

3.1. *Interactive quantities*

The virtual-lab design process includes selecting the *interactive quantities*. These are the model quantities whose value will be interactively changed by the user during the simulation run. The virtual-lab goal is illustrating the dependence between the model dynamic behaviour and the value of these quantities.

Interactive quantities can be parameters, input variables, and time-dependent variables of the *physical model*. For instance, some interactive quantities of the model shown in figure 2 could be the following:

- *Parameters*: the cross-sections of the tank (A) and the outlet hole (a), and the pump parameter (k).
- *Time-dependent variables*: the liquid level (h).
- *Input variables*: the voltage applied to the pump (v).

The *interactive model* combines the dynamic behaviour described in the physical model and the abrupt changes in the value of the interactive quantities produced by the user's actions:

(i) The evolution in time of the interactive *time-dependent quantities* is described by the physical model equations. In addition, their value can change abruptly as a result of the user's interaction.
(ii) The value of the interactive *model parameters* can be abruptly changed by the user's action, remaining constant between consecutive interactive changes.
(iii) The value of the interactive *input variables* is interactively set by the user. Their values change abruptly as a result of the user's action, remaining constant between consecutive changes.

Parameters represent time-independent quantities. Input variables represent boundary conditions which are not calculated from the model equations. Although they are conceptually different, the dynamic behaviour of interactive parameters and interactive input variables is the same. Their values change abruptly

at the interaction instants, remaining constant between consecutive changes. As a consequence, both types of interactive quantities are described in the same manner in the interactive model.

3.2. *Description of the interactive quantities*

In order to support abrupt changes in their values during the simulation run, interactive quantities need to be state variables of the interactive model. The *interactive model* is obtained from the *physical model* by reformulating (when required) the declaration and evaluation of the interactive quantities, so that they become state variables of the interactive model. To this end, the virtual-lab developer has to perform the following tasks.

– *Time-dependent variables* need to be selected as state variables. Modelica 2.0 and Dymola support the user's control on the state variables selection, via the *stateSelect* attribute of Real variables [8, 19–21]. The attribute values include 'never' (the variable will never be selected as state variable) and 'always' (the variable will always be used as a state). If the number of variables selected as state variables by setting the value of their *stateSelect* attribute to 'always' is higher than the model order, then an error message is generated by Dymola. This feature allows the user to select the model state variables without performing any manipulation on the model equations. The required model manipulations are automatically performed by Dymola.
– *Parameters* and *input variables* are redefined as time-dependent variables with zero time-derivative, and they are selected as state variables. For instance, the parameter A of the tank model shown in figure 2 should be a *Real* variable of the interactive model, calculated from the equation $\mathsf{der(A)} = 0$.

```
model tank
  parameter Real Ainitial    "Initial value of the tank section";
  Real A (start = Ainitial)  "Tank section - Interactive quantity";
  ...
equation
  der(A) = 0;
  ...
end tank;
```

Let's consider that all the interactive quantities can be simultaneously selected as state variables (this condition will be removed in section 3.3). Provided that this restriction is satisfied, the virtual-lab developer does not need to perform further modifications in the model. In particular, the required code to implement the user's changes in the value of the interactive quantities is pre-defined in the interactive control elements (shown in figure 1b). Changes in the model state are performed as state re-initialization events by using the Modelica's $reinit(x, expr)$ operator. It re-initializes an state variable (x) with the value obtained by evaluating an expression ($expr$), at the event instant. These changes are triggered using *when* clauses.

Defining the interactive parameters and input variables as state variables increases the number of state variables. This has an unwanted effect: it slows down the simulation. We could think of redefining the interactive parameters and input variables as discrete-time variables or, alternatively, as input variables whose values are provided by the virtual-lab view. In this way, the number of state variables would not be increased. However, this is not a valid approach. Dymola automatically performs model manipulations in order to formulate the model according to the requested state selection. The problem is that these model manipulations can require differentiating an interactive parameter or input variable, which results in an error being generated. An example is discussed next.

Consider the model shown in figure 2. It is formulated according to the state selection $e_1 = \{V\}$. The model can be manipulated as shown below, in order for h to be a state variable instead of V. The variable to evaluate from each equation is written within square brackets.

$$[F] = a\sqrt{2gh} \tag{1}$$

$$[F_{in}] = kv \tag{2}$$

$$[derV] = F_{in} - F \tag{3}$$

$$[V] = Ah \tag{4}$$

$$derV = \dot{A}h + A\left[\dot{h}\right] \tag{5}$$

The time-derivative of the tank cross-section (i.e., $\dot{A}$) appears in Eq. (5). If the interactive quantity A is defined as an input variable, then an error is produced: Dymola can not differentiate an input variable. The same problem arises if A is defined as a discrete-time variable. A valid approach is the previously discussed: defining the interactive parameters and input variables as constant state variables (i.e., $\dot{A} = 0$). The interactive changes in the value of these quantities are implemented by re-initializing their values.

3.3. *Supporting multiple selections of the state variables*

In general, different choices of the state variables are possible. For instance, possible choices in the model shown in figure 2 include:

$$e_1 = \{h\} \qquad e_2 = \{V\} \qquad e_3 = \{F\}$$

where e_i represents one particular choice of the state variables. The state variable selection should be made so that it includes all the interactive quantities. For instance, if the user wants to change interactively the level value (h), the appropriate choice is $e_1 = \{h\}$. Likewise, if the user wants to change the liquid volume, then the right choice is $e_2 = \{V\}$; and if he wants to change the output flow, then $e_3 = \{F\}$.

In addition, changes in the value of interactive parameters can have different effects depending on the state variable choice. Consider a change in the cross-section (A) of the model in figure 2 (for instance, the user interactively changes A from 1 cm^2 to 3 cm^2). If the volume (V) is a state variable, then the change in A produces an abrupt change in the value of the liquid level (h) and flow (F), while the liquid volume remains constant. On the contrary, if the state variable is the height (h) or the flow (F), these quantity values would not change as the result of an instantaneous change in A but the volume would.

In some cases, all interactive quantities can be selected as state variables. Then, the description of the interactive model, as it was proposed in section 3.2, is straightforward. However, this is not always the case. Consider, for instance, a virtual-lab illustrating the dynamic behaviour of the model shown in figure 2. This virtual-lab is required to support two ways of describing the interactive changes in the amount of liquid contained in the tank: changes in the liquid volume (V) and changes in the liquid level (h). Each time the user needs to change the amount of liquid, he can choose between describing it in terms of the volume or in terms of the level. Different choices are possible during a given simulation run. However, V and h can not be simultaneously selected as state variables.

An approach to allow interactive models supporting simultaneously different choices of the state variables is the following. Modelling the interactive model as composed of several instantiations of the physical model, each one with a different choice of the state variables. Modelica capability for state selection control (i.e., *stateSelect* attribute) allows the virtual-lab developer to select the state variables without performing any model manipulation.

Therefore, the interactive model is composed of as many instantiations of the physical model as different state selections are required. The adequate physical-model instantiation (i.e., that with the required state selection) is used for executing each interactive action from the user. That is to say, for performing the abrupt changes in the value of the interactive quantities and for solving the re-start problem. Next, these calculated values are used to re-initialize the other physical-model instantiations. This action guarantees that all physical-model instantiations describe the same trajectory. This procedure is briefly discussed next.

(i) The physical model has to be modified as it was described in section 3.2. In case of the model shown in figure 2, the physical model could be composed of the connection of three components: (1) the pump, modelling the input flow of liquid ($F_{in} = kv$); (2) the tank, describing the conservation of the liquid volume ($dV/dt = F_{in} - F$) and the relationship between the volume and the level of the liquid

($V = Ah$); and (3) the pipe, describing the output flow of liquid ($F = a\sqrt{2gh}$). This physical model could be modified as it is shown next. It is supposed that three selections of the state variables are required: $e_1 = \{h\}$, $e_2 = \{V\}$ and $e_3 = \{F\}$.

```
model tank
  parameter Boolean hIsState  = false;
  parameter Boolean VIsState  = false;
  Real h (stateSelect = if hIsState
                        then StateSelect.always else StateSelect.never) "Liquid level";
  Real V (stateSelect = if VIsState
                        then StateSelect.always else StateSelect.never) "Liquid volume";
  parameter Real Ainitial    "Initial value of the tank section";
  Real A (start = Ainitial)  "Tank section - Interactive quantity";
  ...
equation
  der(A) = 0;
  ...
end tank;

model pipe
  Real F      (stateSelect = if FIsState
                             then StateSelect.always else StateSelect.never)  "Liquid flow";
  parameter Real aInitial = 1  "Initial value of the pipe section";
  Real a (start = aInitial)    "Pipe section - Interactive quantity";
  ...
equation
  der(a) = 0;
  ...
end pipe;

model pump
   parameter Real vInitial   "Initial value of the applied voltage";
   Real v (start = vInitial) "Voltage applied to the pump - Interactive";
   parameter Real kInitial   "Initial value of the pump parameter";
   Real k (start = kInitial) "Pump parameter  - Interactive quantity";
   ...
equation
   der(v) = 0;
   der(k) = 0;
   ...
end pump;

partial model physicalModel
  parameter Boolean[3] isState;
  tank tank1 ( hIsState  = isState[1],
               VIsState  = isState[2], ...);
  pipe pipe1 ( FIsState  = isState[3], ...);
  pump pump1 ( ... );
  ...
end physicalModel;
```

The boolean vector isState[:], declared in physicalModel, allows controlling the state selection. The size of this vector is equal to the number of interactive time-dependent quantities. For instance, if isState[:] is set to the value {false,true,false} when instantiating the physical model, then the liquid volume (V) is selected as a state variable. Also, the interactive parameters (A, a) and the input variable (v) have been defined as constant state variables. This first step in the implementation of the interactive model is represented in figure 3a.

(ii) The setParamVar class is defined (see figure 3b). It inherits from physicalModel, and it contains the *when*-clauses required to change the value of the interactive parameters and input variables. These interactive quantities are represented by the ivars[:] array, and their new values, specified interactively

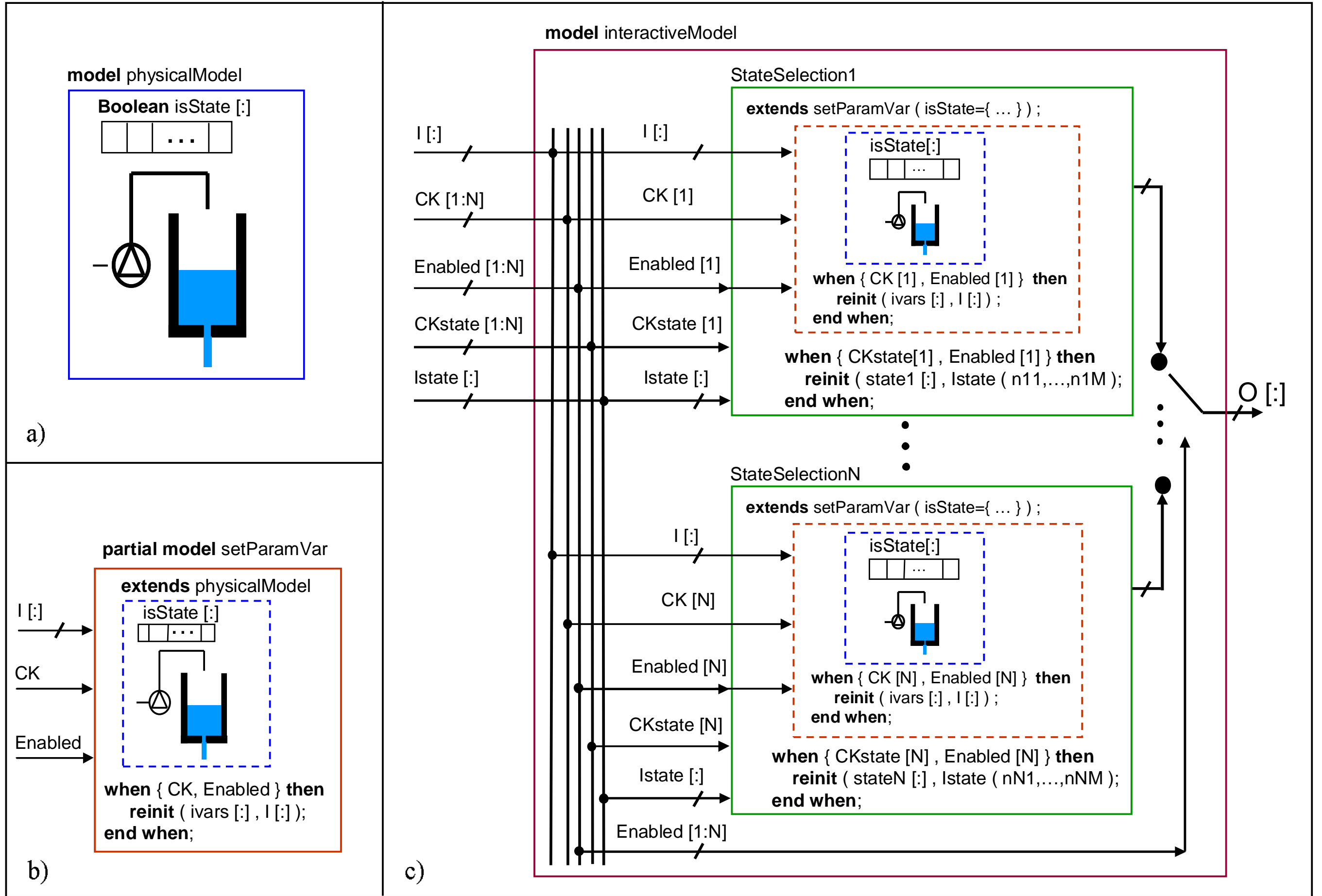

Figure 3. Schematic description of the proposed modelling methodology for interactive simulation.

by the virtual-lab user, are represented by the I[:] array. The size of these arrays is equal to the number of interactive parameters plus the number of interactive input variables. The *when*-clauses are triggered by the boolean variables CK and Enabled. When the value of any of these two variables changes from *false* to *true*, then the ivars[:] array is re-initialized to the value of the I[:] array.

(iii) There are defined as many components (StateSelection1, …, StateSelectionN) as different state-variable choices are required (e_1 = state1[:], …, e_N = stateN[:]). The number of state-variable choices to be supported by the virtual-lab is represented by N. The class of these components inherits from setParamVar (see figure 3c). In addition, it contains the *when*-clauses required to re-initialize its state-variable array (i.e., state array) to the values interactively set by the user (i.e., Istate array).

The CK[1:N] and Enabled[1:N] arrays trigger the re-initialization of the interactive parameters and input variables. The CKstate[1:N] and Enabled[1:N] arrays trigger the re-initialization of the interactive time-dependent quantities. The $i-th$ component of these arrays controls the $i-th$ instantiation of the physical system (i.e., StateSelection$i$).

The array Enabled[1:N] indicates which state-variable selection is enabled. It is used to select which output is connected to the output variables (O[:]). These are the variables used to refresh the virtual-lab view.

4. VirtualLabBuilder library

VirtualLabBuilder library is composed of the packages shown in figure 1a. Some of them are intended to be used by the virtual-lab developers (i.e., *VirtualLabBuilder* users). These are: (1) ViewElements and VLabModels packages, which contain the classes required to implement the virtual-lab view and to set up the complete virtual-lab; and (2) Examples package, which contains some tutorial material illustrating the library use. The documentation of these packages is oriented to the *VirtualLabBuilder* users.

On the contrary, the classes within the src package are not intended to be directly used by the virtual-lab developers. The documentation of this package describes the implementation details required to modify and extend the *VirtualLabBuilder* library. In fact, the classes within ViewElements and VLabModels packages inherit from classes defined within src package, inheriting the structure and the behaviour, and adding only the documentation oriented to the virtual-lab developer.

VLabModels package contains two classes: PartialView and VirtualLab. The purpose of PartialView and VirtualLab classes was briefly described in section 2. They have to be the super-classes of the models defining the virtual-lab view and the complete virtual-lab respectively. Further details can be found in the *VirtualLabBuilder* library documentation. ViewElements package is discussed next.

4.1. *Interactive graphic elements*

ViewElements package contains the graphic elements that can be used to define the view. The *initial sections* of these elements contain calls to Modelica functions that perform calls to external C-functions. These C-functions write the Java code of the elements to a file, generating automatically the Java application (i.e., a *.jar* file) that is the virtual-lab view. The three packages included within ViewElements are briefly described next. A detailed description of these graphic elements and their properties can be found in [14].

- Containers package has those graphic elements that are intended to host other graphic elements. The container properties are set in the view definition and they can not be modified during the simulation run. *VirtualLabBuilder* contains the following five classes of containers: MainFrame, Dialog, Panel, DrawingPanel and PlottingPanel (see figure 1d).
- Drawables package contains several classes implementing interactive 2-D shapes, whose properties (i.e., size, position, rotation angle, aspect ratio, colour, etc.) can be linked to the model variables. They are intended to be used for building animated and interactive schematic representations of the system. These classes are: Polygon, Oval, Text, Arrow and Trail (see figure 1c). Objects of Drawables classes must be placed inside containers that provide a coordinate system (i.e., containers of DrawingPanel and PlottingPanel classes).

  In addition to these general-purpose interactive components, other domain-specific components can be implemented. In order to demonstrate this capability, Mechanics package has been included within Drawables package. It contains two classes (i.e., Damper and Spring) implementing an interactive damper and an interactive spring.
- InteractiveControls package contains classes that allow modifying interactively the value of model variables. These are: Slider, NumberField, RadioButton, Button1Action, Button2Actions, Label and Checkbox. In addition, the PauseButton class creates a button that allows the user to pause and resume the simulation by clicking on it. The InfoButton class creates a button that allows the user to show and hide a window displaying HTML pages. This feature allows including documentation in the virtual-lab. That is to say, it supports the implementation of the *virtual-lab introduction.*

4.2. *Connection rules*

The interface of the interactive graphic components is composed of connectors, which facilitate the connection among the components. Four connector types have been defined. Each one has a distinctive icon. Connector icons are square or circular, empty or filled. The following two types of interfaces have been defined (see figures 1b, 1c & 1d):

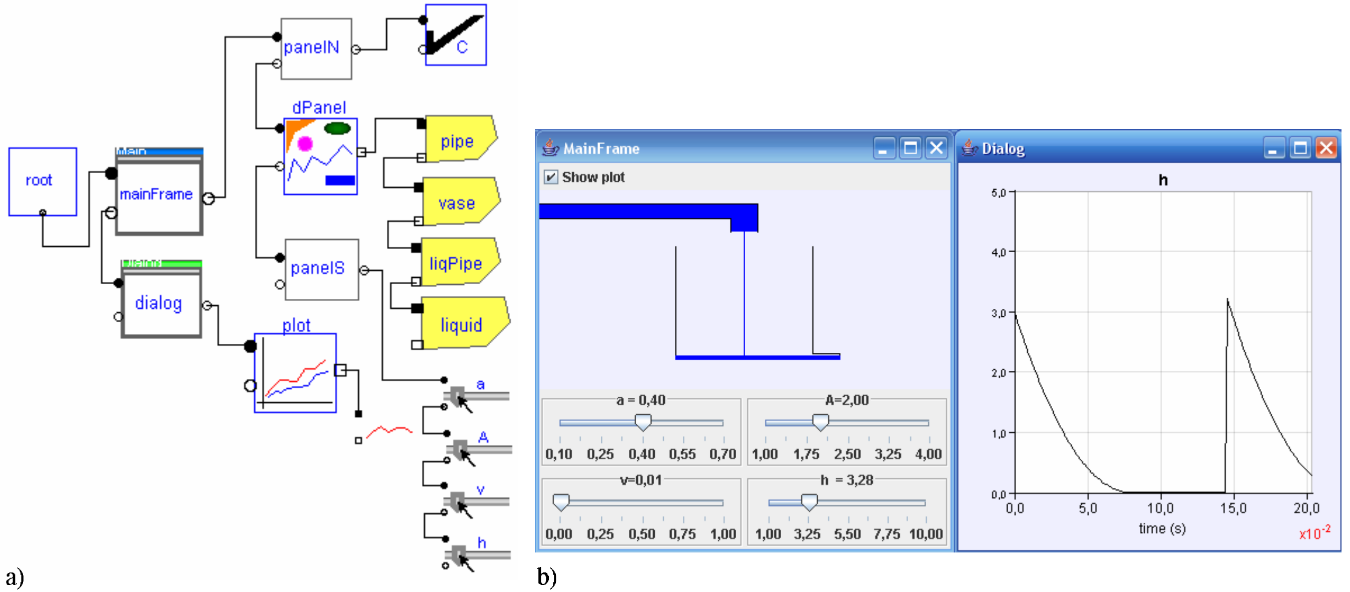

Figure 4. Tank process: a) Modelica description of the virtual-lab view; and b) virtual-lab.

– *Interface of container components.* It has three connectors (see figure 1d). Two placed on one side (called 'left connectors') and the third one (called 'right connector') placed on the opposite side.
– *Interface of interactive controls and drawable elements.* It has two connectors (called 'left connectors'): one filled and one empty (see figures 1b & 1c).

The virtual-lab programmer must observe the following three rules when connecting the graphic elements:

(i) Only connectors with the same shape (circular or square) can be connected.
(ii) Each filled connector must be connected to one and only one empty connector.
(iii) Each empty connector can be left unconnected or can be connected to one and only one filled connector.

The meaning of the connections among the graphic components is as follows:

– If two components are connected using their 'left connectors', then both components are hosted within the same container. The component position in the chain of connected elements determines its insertion order within the container.
– If two components are connected using the 'right connector' of the first component and a 'left connector' of the second component, then the second component is hosted within the first component.

The following example tries to illustrate how the graphic elements can be used to compose the view of a virtual-lab. In particular, the view of the tank process described in section 3. The Modelica description of the virtual-lab view and the obtained virtual-lab are shown in figure 4. In this case, the model of the tank process has only one state selection and one state variable (the liquid level).

The mainFrame and dialog components are hosted inside root. The dPanel, panelS and panelN components are hosted inside mainFrame. The C component is hosted inside panelN. The pipe, vase, liqPipe and liquid components are hosted inside dPanel. The a, A, v and h components are hosted inside panelS. The plot component is hosted inside dialog. Finally, the component trail of Trail class is hosted inside plot.

The window showing the component parameters is displayed by double clicking on the component icon. The parameter windows of the components trail, a and mainFrame are shown in figures 5a, b & c respectively.

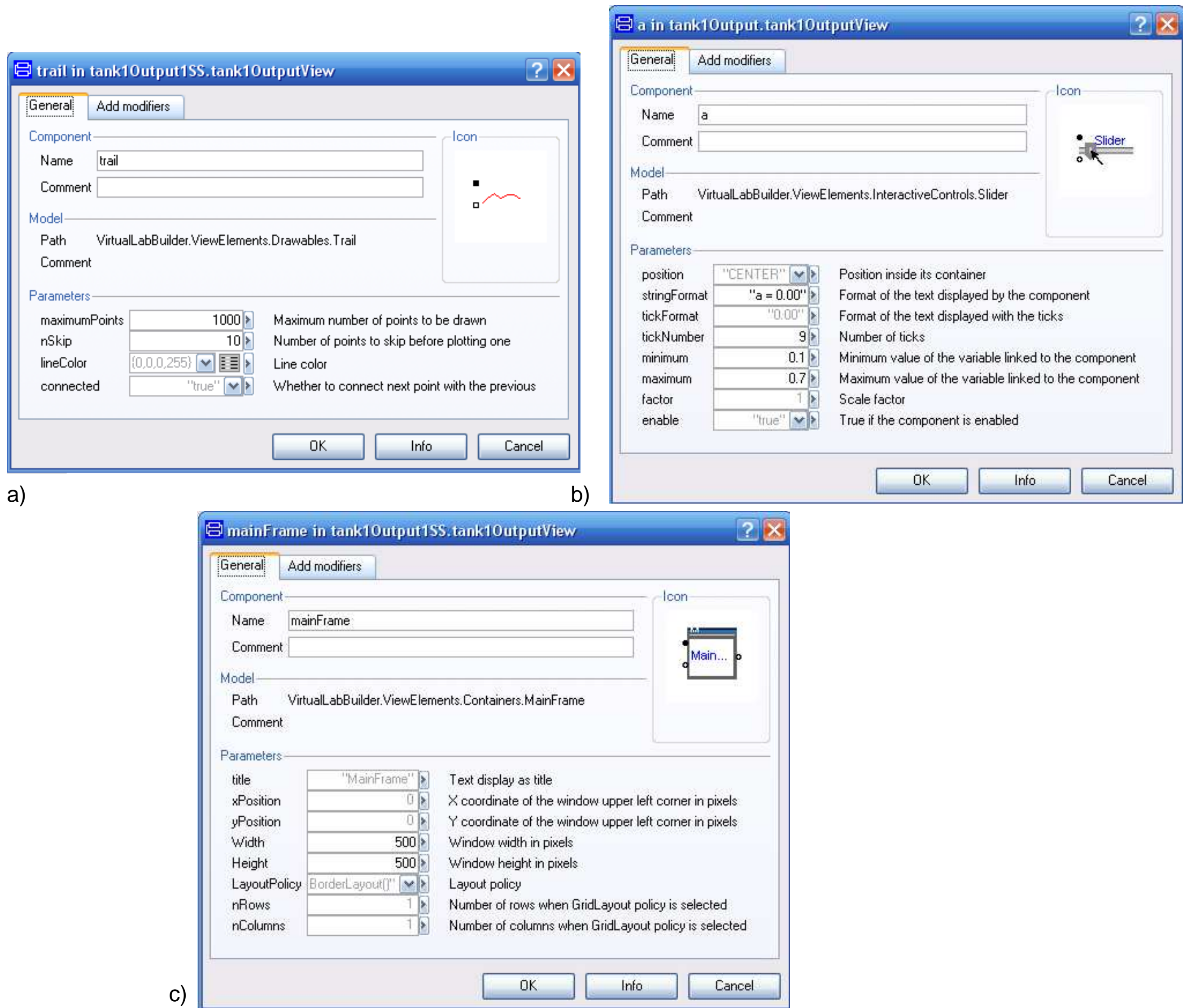

Figure 5. Parameter window of the components: a) trail; b) a; and c) mainFrame.

5. Case study: virtual-lab of a solar house

The implementation of a virtual-lab intended to illustrate the thermodynamics of a solar house is discussed. This virtual-lab allows the user to: (1) change the thermodynamic properties of the slab, the outer and inner walls, and the roof; (2) turn on and off the air conditioning, which is placed in the living room; and (3) set the parameters of the air conditioning control system (i.e., the setpoints for the minimum and maximum values of the temperature).

The virtual-lab view contains the floor plan of the house (see figure 8b). The room colours change between blue and red as a function of the temperature inside the room. The heat flow through the outer walls are represented by arrows. The width and orientation of the arrow are functions of the magnitude and the direction of the heat flow, respectively. Also, the virtual-lab view contains plots of some selected quantities (see figure 11).

5.1. *The Modelica model of the solar house*

This solar house model is included within the Bondlib library [22]. The model describes the thermodynamic behaviour of an experimental house located near the airport in Tucson, Arizona, with a passive solar heating

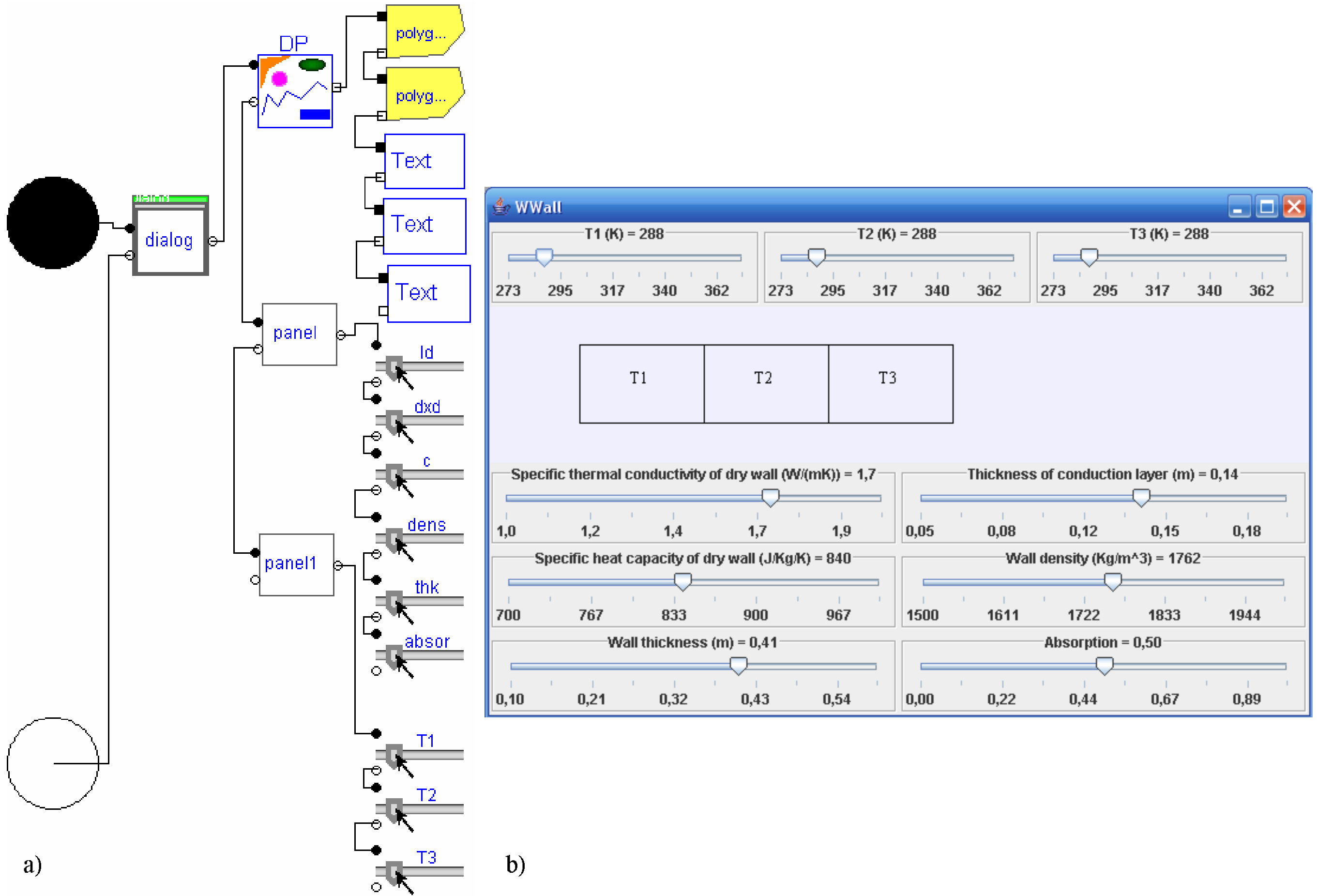

Figure 6. ExWallView class: a) diagram; and b) generated view.

system. The house has four rooms: two bedrooms, a living room and a solarium that collects heat during the winter and releases it during the summer. The living room has an air conditioning unit.

The four rooms are composed using models that describe the outer and inner walls, the roofs, the windows and the slabs. The bond graph technique is used to model the physical laws of heat transfer between the basic components of the house, regarding conduction, convection and radiation. A detailed description of the model can be found in [23, 24].

5.2. *Composing the virtual-lab*

The solar house model has been adapted to suit interactive simulation. Interactive parameters and input variables have been re-defined as constant state variables (i.e., with zero time-derivative).

The Modelica description of the virtual-lab view has been developed modularly, by extending and connecting the required graphic components of the *VirtualLabBuilder* library. Modelica classes have been programmed to describe the view associated to an inner wall (InWallView), an outer wall (ExWallView), a slab (SlabView) and a roof (RoofView). These are described next:

– The diagram of the ExWallView Modelica class is shown in figure 6a and the graphic interface generated is shown in figure 6b. The ExWallView class contains instances of graphic elements contained in *VirtualLabBuilder* library (i.e., Dialog, DrawinPanel, Panel, Polygon, Text and Slider). The connection among these elements determines the layout of the graphic interface. The graphic interface consists of a window that contains a set of sliders at the bottom and at the top (see figure 6b). These sliders allow the user to modify the temperature and thermodynamic properties of the wall, including the specific thermal con-

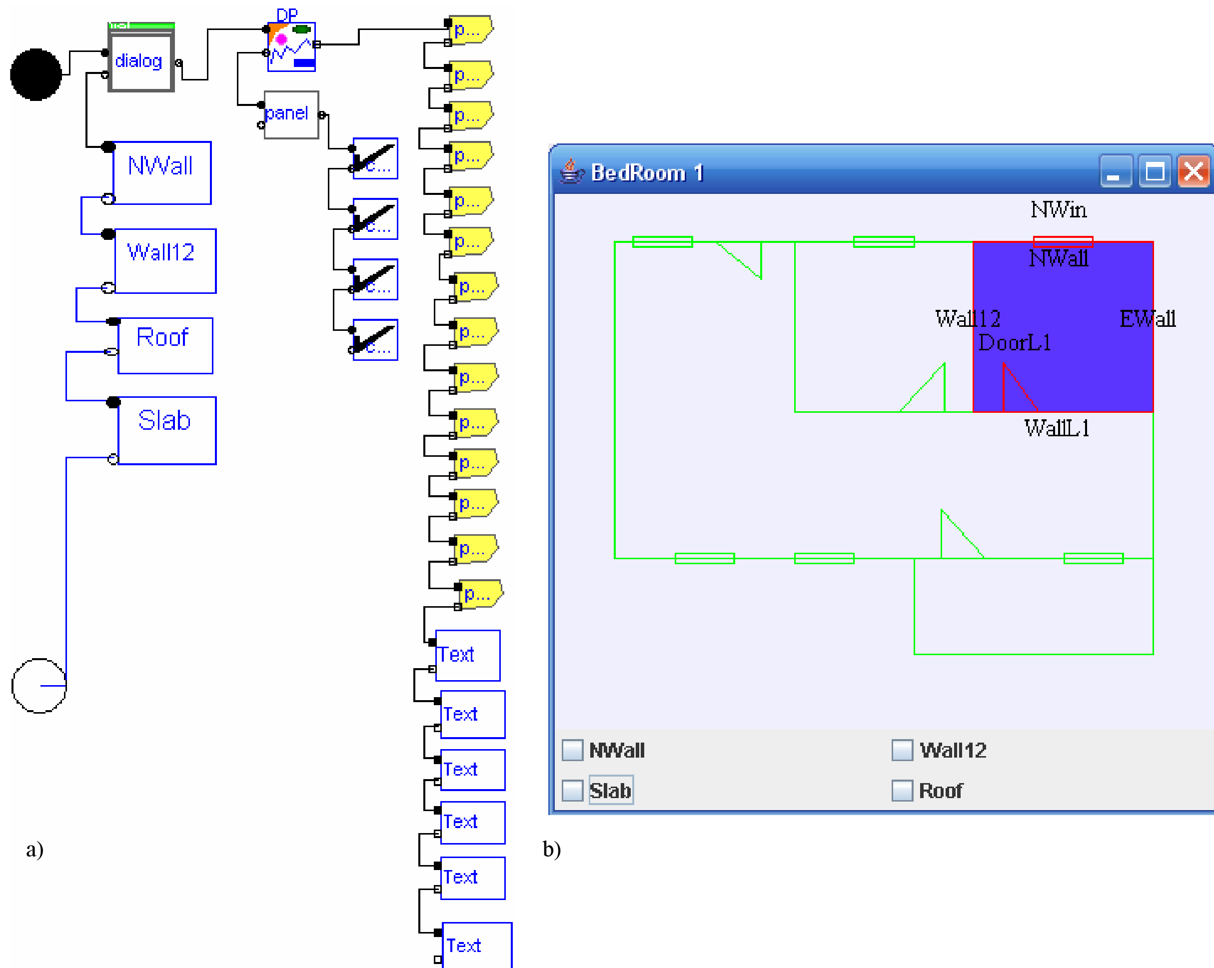

Figure 7. BedRoom1View class: a) diagram; and b) generated view.

ductivity of the dry wall, the thickness of the conduction layer, the specific heat capacity, the density, the thickness of the outer wall, and absorption coefficient. The centre of the window contains a graphical representation of the wall model, which is composed of three conducting layers.

– InWallView class contains sliders that allow the user to change the temperature and thermodynamic properties of the wall (i.e., specific thermal conductivity of the dry wall, thickness of the conduction layer, specific heat capacity, density and thickness).
– RoofView class contains sliders that allow the user to change the thermodynamic properties (i.e., specific thermal conductivity, thickness, specific heat capacity and density) of the three conducting layers that compose the roof.
– SlabView class contains sliders that allow the user to change the slab thermodynamic properties (i.e., specific thermal conductivity, thickness of the slab, specific heat capacity, density and thickness of the conduction layer).

Modelica classes have been composed using *VirtualLabBuilder* to describe the view associated to the house (HouseView), the living room (LivingRoomView), and bedrooms 1 and 2 (BedRoom1View and BedRoom2View). These are briefly described next:

– The diagram of the BedRoom1View class is shown in figure 7a, and the generated graphic interface is shown in figure 7b. This model contains instances of SlabView, RoofView, ExWallView and InWallView classes. The view consists of a window that has a set of checkboxes at the bottom and the floor plan of the room at the centre (see figure 7b). The checkboxes allow the user to show and hide the windows associated to each building component of the room (outer and inner walls, slab and roof).
– The diagram of the HouseView class is shown in figure 8a, and the generated graphic interface is shown in

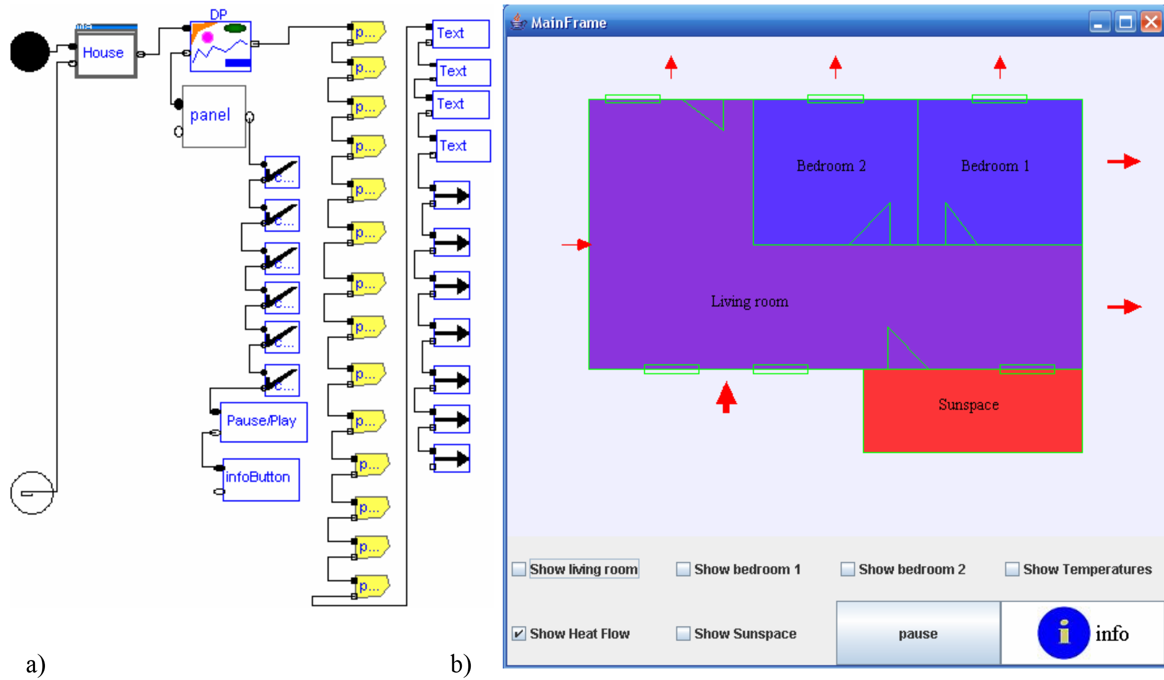

Figure 8. HouseView class: a) diagram; and b) generated view.

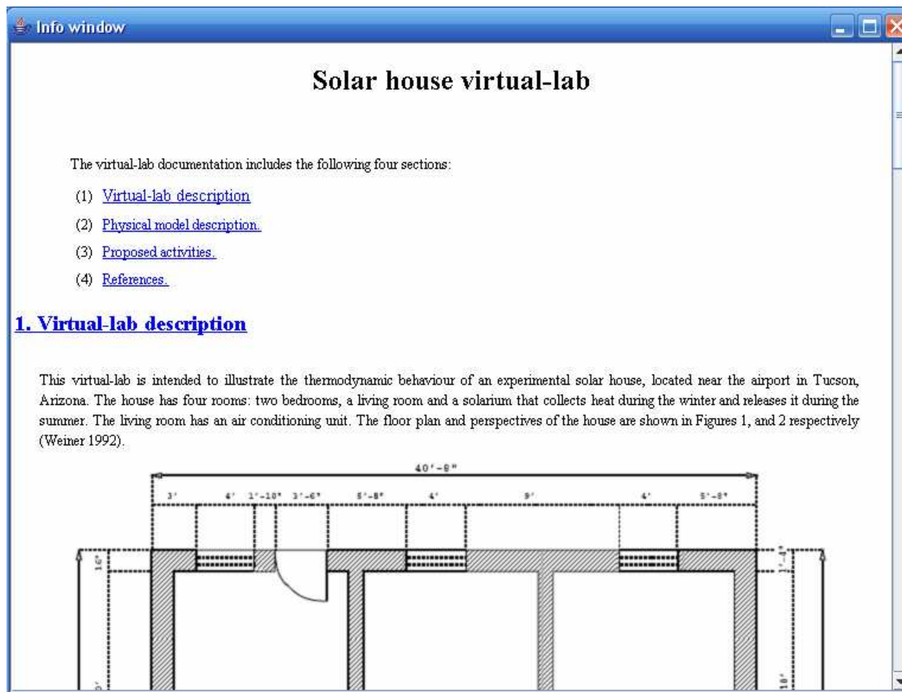

Figure 9. Introduction of the solar house virtual-lab.

figure 8b. The view consists of a window that has a set of checkboxes and two buttons at the bottom and a diagram of the house floor plan in the centre (see figure 8b). The checkboxes allow the user to show and hide the windows associated to the bedrooms 1 and 2, and to the living room. The two buttons allow the user: (1) to pause and resume the simulation; and (2) to show and hide the virtual-lab documentation (see figure 9). Each room of the floor plan has a colour, that change from blue to red depending on the room temperature. The arrows shown in the floor plan represent the heat flow through the outer walls (see figure 8b). The width and orientation of the arrows depend on the magnitude and the direction of the heat flow, respectively.

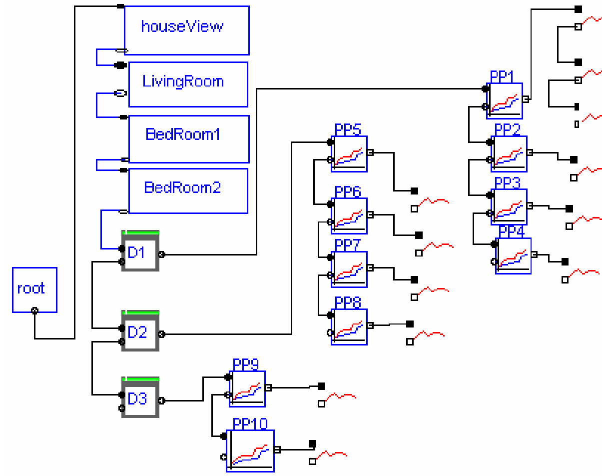

Figure 10. Modelica diagram of the complete virtual-lab view.

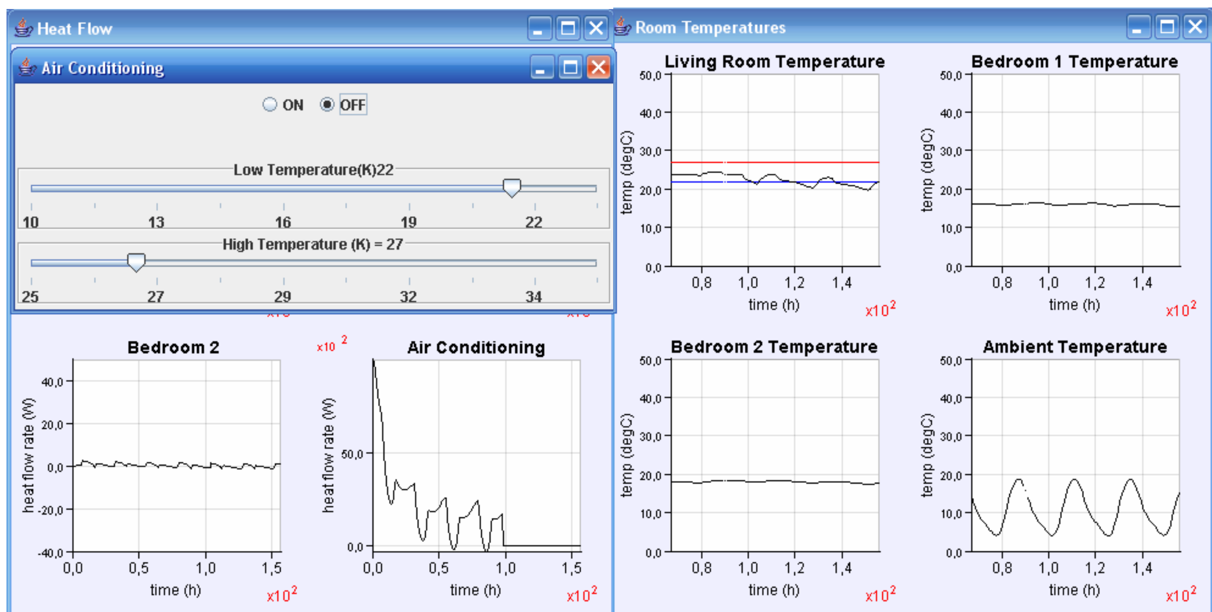

Figure 11. Dynamic response of some selected quantities.

The Modelica description of the complete view (i.e, class View) is shown in figure 10. This model extends the PartialView class and contains instances of the *VirtualLabBuilder* library components describing plots. These plots are used to display the time evolution of the heat flow and the temperature in the rooms of the house.

5.3. *Virtual-lab set up and launch*

The virtual-lab description is obtained as discussed in section 3. It is translated using Dymola and executed. Then, the jar file containing the Java code of the virtual-lab view is automatically generated and executed.

The dynamic response of the solar house when the air conditioning is turned off is shown in figure 11. This change has been interactively performed by the virtual-lab user at the simulated time 100 h. The following six plots are shown in figure 11: (1) the heat flow rate in bedroom 2; (2) the heat flow rate of the air conditioning; (3) the living room temperature and the setpoint value for the minimum and maximum temperatures; (4) the bedroom 1 temperature; (5) the bedroom 2 temperature; and (6) the ambient temperature.

6. Conclusions

A novel approach to virtual-lab implementation using Modelica language has been proposed and it has been successfully applied. The proposed approach has several advantages. Firstly, the virtual-lab is completely described using Modelica language, an object-oriented modelling language aimed to be a de facto standard for representing models and to support model exchange. Secondly, existing Modelica libraries for modelling of physical systems can be reused in order to build the virtual-lab models. Finally, the virtual-lab view is modelled using the object-oriented paradigm, which facilitates its development, maintenance and reuse.

In order to support the application of this approach, the following two tasks have been completed: (1) the proposal of a modelling methodology intended to transform any Modelica model into a description suitable for interactive simulation; and (2) the design and implementation of a Modelica library, named *VirtualLabBuilder*, supporting the description of the virtual-lab view and the bi-directional communication between the model and the view.

This modelling methodology, and the architecture and use of *VirtualLabBuilder* have been discussed. The proposed approach has been illustrated by means of a case study: the virtual-lab describing the thermodynamic behaviour of a solar house. The model describing the solar-house has been adapted to suit the interactive simulation. The virtual-lab view has been implemented using *VirtualLabBuilder*.

Acknowledgements

This work has been supported by the Spanish CICYT, under DPI2004-01804 grant, and by the IV PRICIT (Plan Regional de Ciencia y Tecnologia de la Comunidad de Madrid, 2005-2008), under S-0505/DPI/0391 grant.

References

[1] Dormido, S., 2004, Control learning: Present and Future. *Annual Reviews in Control,* **28**, 115–136.
[2] Euler, M. and Müller, A., 1999, Physics learning and the computer: A review, with a taste of meta-analysis. In: *Proceedings of the* 2nd *International Conference of the European Science Education Research Association*, Kiel, Germany.
[3] Esquembre, F., 2004, Easy Java Simulations: a Software Tool to Create Scientific Simulations in Java. *Computer Physics Communications,* **156**, 199–204.
[4] Easy Java Simulations website: http://fem.um.es/Ejs/
[5] Sysquake website: http://www.calerga.com/
[6] Modelica Association website: http://www.modelica.org/
[7] Åström, K., Elmqvist, H. and Mattsson, S. E., 1998, Evolution of Continuous-Time Modeling and Simulation. In: *Proceedings of the* 12th *European Simulation Multiconference*, Manchester, UK, pp. 9–18.
[8] Dynasim AB, 2006, *Dymola. User's Manual*, Lund, Sweden.
[9] The OpenModelica Project website: http://www.ida.liu.se/ pelab/modelica/OpenModelica.html/
[10] Engelson, V., 2000, Tools for Design, Interactive Simulation, and Visualization of Object-Oriented Models in Scientific Computing. PhD Thesis, Linkping University, Sweden.
[11] Martin, C., Urquia, A., Sanchez, J., Dormido, S., Esquembre, F., Guzman, J.L. and Berenguel, M., 2004, Interactive Simulation of Object-Oriented Hybrid Models, by Combined use of Ejs, Matlab/Simulink and Modelica/Dymola. In: *Proceedings of the* 18th *European Simulation Multiconference*, Magdeburg, Germany, pp. 210–215.
[12] Martin, C., Urquia, A. and Dormido, S., 2005, Object-oriented modelling of interactive virtual laboratories with Modelica. In: *Proceedings of the* 4th *International Modelica Conference*, Hamburg, Germany, pp. 159-168.
[13] Martin, C., Urquia, A. and Dormido, S., 2005, Object-Oriented Modeling of Virtual Laboratories for Control Education. In: *Proceedings of the* 16th *IFAC World Congress*, Prague, Czech Republic, paper code Th-A22-TO/2.
[14] Martin, C., Urquia, A. and Dormido, S., 2006, An Approach to Virtual-lab Implementation using Modelica, In: *Proceedings of the* 20th *Annual European Simulation and Modelling Conference*, Toulouse, France, pp. 137–141.
[15] Martin, C., Urquia, A. and Dormido, S., Implementation of Interactive Virtual Laboratories for Control Education Using Modelica, 2007, In: *Proceedings of European Control Conference 2007*, Kos, Greece, pp. 2679–2686.
[19] Otter, M. and Olsson, H., 2002, New features in Modelica 2.0. In: *Proceedings of the* 2nd *International Modelica Conference*, Oberpfaffenhofen, Germany, pp. 7.1–7.12.
[20] Mattsson, S.E., Olsson, H. and Elmqvist, H., 2000, Dynamic Selection of States in Dymola. In: *Proceedings of the Modelica Workshop*, Lund, Sweden, pp. 61–67.
[21] Fritzson, P., 2004, *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, IEEE Press - Wiley, John & Sons.
[22] Cellier, F.E. and Nebot, A., 2006, The Modelica Bond-Graph Library. In: *Proceedings of the* 4th *International Modelica Conference*, Hamburg, Germany, pp. 57–65.
[23] Weiner, M., 1992, Bond Graph Model of a Passive Solar Heating System. MS Thesis, University of Arizona, Tucson, Arizona.
[24] Weiner, M. and Cellier, F.E., 1993, Modeling and Simulation of a Solar Energy System by Use of Bond Graphs. In: *Proceedings of the* 1st *SCS International Conference on Bond Graph Modeling*, San Diego, California, pp. 301–306.