Abstract—An approach to the implementation of virtual-labs using Modelica language is discussed. Firstly, a systematic methodology to transform any Modelica model into a formulation suitable for interactive simulation has been proposed. Secondly, a Modelica library, called VirtualLabBuilder, has been designed and programmed. This library is intended to facilitate the implementation of interactive visual interfaces for Modelica models. VirtualLabBuilder contains a set of ready-to-use Modelica models of visual interactive elements (i.e., containers, animated geometric shapes and interactive controls), that allows easy creation of the virtual-lab view.

The application of this approach to the implementation of a virtual-lab describing the thermodynamic behaviour of a solar house is discussed in this manuscript. The solar house model was developed by Markus Weiner as a part of his M.S. thesis and it was included in the BondLib Modelica library by F.E. Cellier. This Modelica model has been adapted for interactive simulation by using the proposed methodology. The interactive graphic user-to-model interface has been built by using VirtualLabBuilder. The virtual-lab obtained is completely written in Modelica language.

Index Terms—Control education, interactive simulation, Modelica, object-oriented modeling.

I. INTRODUCTION

Virtual-labs supporting interactive simulations are effective pedagogical resources for control education [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 to 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 a model and a view. The model is the mathematical model representing the relevant behavior 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 behavior 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.

The implementation of virtual-labs using Modelica language [2] has been proposed in [3]. To achieve this goal, the following two tasks have been completed [3]:

1) 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.
2) VirtualLabBuilder Modelica library has been designed and programmed. It includes Modelica models implementing graphic interactive elements, such as containers, animated geometric shapes (polygon, arrow and oval) and interactive controls (checkbox, slider and radio-button). 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 communication between the view and the model.

A. Contributions of this Paper

The architecture and use of VirtualLabBuilder is discussed, and the library is applied to the implementation of a virtual-lab describing the thermodynamic behaviour of a solar house. The solar house model was developed by Markus Weiner as a part of his M.S. thesis [4, 5] and it was included in the BondLib Modelica library [6] by F.E. Cellier. This model has been adapted for interactive simulation and the virtual-lab view has been composed by using VirtualLabBuilder. As a result, the feasibility of setting up virtual-labs of complex Modelica models by using VirtualLabBuilder has been demonstrated.

The architecture of VirtualLabBuilder and the procedure to implement virtual-labs using this library are discussed in Sections II and III. Finally, the implementation of the solar house virtual-lab is discussed in Section IV.

II. VIRTUAL-LAB IMPLEMENTATION USING VirtualLabBuilder LIBRARY

The virtual-lab definition includes the description of the model, the view and the bidirectional flow of information between the model and the view. The steps to define the virtual-lab according to the proposed methodology are discussed next.
A. Description of the Virtual-Lab Model

The Modelica model has to be adapted to suit interactive simulation following the methodology described in [3]. Basically, the parameters and input variables need to be declared as constant state variables (i.e., with zero time-derivative) in order to make them interactive.

B. Description of the Virtual-Lab View

VirtualLabBuilder is composed of the following five packages (see Figure 1a):

- The ViewModel package contains the PartialView and the VirtualLab classes. As it will be discussed later, these classes are used to define the virtual-lab view and the complete virtual-lab respectively.
- The ViewElements package contains the graphic interactive elements that the user can employ to compose the virtual-lab view. The content of this package is shown in Figure 1b and it will be described in Section III.
- The Interfaces, Functions and TypesDef packages contain classes used by other classes of the library. These packages are not directly used by the virtual-lab developers.

The Modelica class describing the view must be a subclass of PartialView class. This class contains the code required to perform the communication between the model and the view. This code is valid for any model and view descriptions, and the virtual-lab designer only needs to set the length of the model-to-view communication interval.

The graphic components have to be connected by the virtual-lab programmer. The “root” graphic component (i.e., the container component which hosts the rest of the components), named root, is pre-defined in the PartialView class. The connections among the graphic components determine their layout in the virtual-lab view. Modelica modeling environments (i.e., Dymola [7], OpenModelica [8], etc.) allow defining in a drag-and-drop way the instantiation of the required VirtualLabBuilder library components and connecting them using the mouse.

C. Virtual-Lab Set Up

The Modelica description of the virtual-lab has to be an instance of VirtualLab class. This class contains two parameterized generic classes: the class of the virtual-lab model and the class of the view. In addition, the virtual-lab designer has to specify, writing the required equations, how the variables of the model and the view classes are connected.

D. Translation to Executable Code and Launch

The virtual-lab description, obtained as discussed in Section II-C, is translated using Dymola and ran. As a part of the model initialization (i.e., the calculations performed to find the initial value of the model variables), the initial sections of the interactive graphic objects 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 and packed into a single jar file. Also, the communication procedure between the model and the view is set up. This communication is based on client-server architecture: the C-program generated by Dymola [7] is the server and the Java program (which is automatically generated during the model initialization) is the client.

Once the jar file has been created, it has to be executed by the virtual-lab user. As a result, 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:

- The model simulation (i.e., the server) sends to the view (i.e., the client) the data required to refresh the view.
- The view sends to the model simulation the new value of the variables modified due to the user’s interactive action.

III. INTERACTIVE GRAPHIC ELEMENTS

The ViewElements package contains the graphic elements that can be used to define the view. These elements (see Figure 1b) can be classified into the following three categories:

- Containers (MainFrame, Dialog, Panel, DrawingPanel and PlottingPanel classes). These graphic elements can host other graphic elements. The properties of these elements are set as a part of the view definition and they can not be modified during the interactive simulation run.
- Drawables (Polygon, Oval, Text, Arrow and Trail classes). These elements can be used to build an animated
schematic representation of the system. The variables setting the geometric properties of these elements (position, size, etc.) can be linked to model variables.

- Interactive controls (Slider, Label, RadioButton and CheckBox classes). Model variables can be linked to the variables defining the state of the interactive control elements. This allows the user to change the value of these model variables during the simulation run.

Drawable elements and interactive controls implement the information flow between the model and the view of the virtual-lab. The simulated value of the model variables modifies the properties of the drawable elements (i.e., model-to-view information flow). The user’s interactive action on the interactive controls modifies the value of the model variables (i.e., view-to-model information flow).

IV. SOLAR HOUSE VIRTUAL-LAB

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 4b). The room colors 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 magnitudes (see Figure 6).

A. THE MODELICA MODEL OF THE SOLAR HOUSE

This solar house model is included within the Bondlib library [6]. The model describes the thermodynamic behaviour of an experimental house located near the airport in Tucson, Arizona, with a passive solar heating 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.

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 [4, 5].

B. COMPOSING THE VIRTUAL-LAB

The solar house model has been adapted to suit interactive simulation. Interactive parameters and user-controlled 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 (see Figure 1b). 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:

- ExWallView class is shown in Figure 2a and the graphic interface generated is shown in Figure 2b. 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 determine the layout of the graphic interface. The graphic interface consists of a window that contains a set of sliders at the bottom and the top (see Figure 2b). These sliders allow the user to modify the wall temperature and its thermodynamic properties (i.e., specific thermal conductivity of the dry wall, thickness of the conduction layer, specific heat capacity, density, thickness of the outer wall and absorption coefficient).
Fig. 3. **BedRoom1View** class: a) diagram of the Modelica description; and b) generated view.

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 wall temperature and its thermodynamic properties (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 its 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 programmed 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:

- **BedRoom1View** class is shown in Figure 3a and the graphic interface generated is shown in Figure 3b. 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 3b). 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).
- **HouseView** class is shown in Figure 4a and the graphic interface generated is shown in Figure 4b. The view consists of a window that has a set of checkboxes at the bottom and a diagram of the house floor plan in
Fig. 5. Modelica diagram of the complete virtual-lab view.

The checkboxes allow the user to show and hide the windows associated to the bedrooms 1 and 2, and to the living room. Each room of the plan has a color, that change from blue to red depending on the room temperature. The arrows shown in the plan represent the heat flow through the outer walls (see Figure 4b). The width and orientation of the arrows depend on the magnitude and the direction of the heat flow, respectively.

The Modelica description of the complete view (i.e, class View) is shown in Figure 5. This model extends the PartialView class, which contains: a) one pre-defined graphic element: root; and b) the code required to perform the communication between the model and the view. The View class contains instances of BedRoom1View, BedRoom2View and LivingRoomView classes. It also 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.

The Modelica description of the virtual-lab has to be an instance of VirtualLab class. This class contains: a) two parametrized generic classes: the classes of the virtual-lab model and view; and b) the equations that link the variables of the model and the view classes.

C. Virtual-Lab Launch

The Modelica description of the virtual-lab is translated using Dymola and ran. Then, the jar file containing the Java code of the virtual-lab view is automatically generated. When the virtual-lab user executes this jar file, the virtual-lab view is displayed and the client-server communication is established.

Then, the model response starts. During the simulation run, there is a bi-directional flow of information between the model and the view.

The dynamic response of the solar house when the air conditioning is turned off is shown in Figure 6. 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 6: (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.

V. Conclusions

The feasibility of setting up virtual-labs of complex Modelica models by using VirtualLabBuilder has been demonstrated.
This approach has two strong points. Firstly, the virtual-lab is completely described using Modelica language, an object-oriented modeling language aimed to be a de-facto standard for representing models and to support model exchange. Secondly, VirtualLabBuilder library allows performing an object-oriented description of the virtual-lab view, which facilitates its development, maintenance and reuse.

The architecture of VirtualLabBuilder and how to use it in order to implement virtual-labs has been discussed. Finally, VirtualLabBuilder has been used to implement a virtual-lab describing the thermodynamic behaviour of a solar house. The model describing the solar house has been adapted to suit interactive simulation. The view has been implemented using graphic elements of VirtualLabBuilder.

ACKNOWLEDGMENTS

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