Received May 19, 2022, accepted May 27, 2022, date of publication June 2, 2022, date of current version June 8, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3179712

An Approach to Develop Collaborative Virtual Labs in Modelica

CARLA MARTIN-VILLALBA AND ALFONSO URQUIA
Departamento de Informática y Automática, Universidad Nacional de Educación a Distancia (UNED), 28040 Madrid, Spain
Corresponding author: Carla Martin-Villalba (carla@dia.uned.es)

This work was supported in part by the Ministerio de Economía y Competitividad of Spain under Grant DPI2013-42941-R, and in part by the Vicerrectorado de Investigación e Internacionalización of Universidad Nacional de Educación a Distancia (UNED) for Dissemination of Research Results (Innovation in Education) under Project GID2016-04.

ABSTRACT Virtual labs are valuable educational resources in control education, and are widely used in the process industry as tools for operator training and decision aid. In these application domains, virtual labs typically rely on the interactive simulation of large-scale hybrid-DAE models with components of different engineering domains, whose description can be greatly simplified by the use of the Modelica language. Existing free and commercial Modelica libraries of different domains can be used to describe these models. The Interactive Modelica library facilitates developing virtual labs based on Modelica models, using only Modelica. A new major release of the Interactive Modelica library is presented in this paper, whose most relevant feature is to facilitate the implementation of collaborative virtual labs written using only the Modelica language. This library can be used with the environment OpenModelica, facilitating the implementation of cooperative virtual labs using only open software. This type of virtual lab, which allows several students to interact cooperatively with the same model simulation run, is an effective tool in the context of collaborative learning methods. The efficient communication among the graphical user interfaces and the simulation model is a key issue. We developed a new communication protocol, a synchronization algorithm, and redesigned the Modelica classes of the library to make the communication completely transparent to virtual lab developers. The implementation of a collaborative virtual lab for process control education, based on a simplified version of the Tennessee Eastman process, is discussed. The Interactive Modelica library is freely distributed under Modelica License 2 and can be downloaded from http://www.euclides.dia.uned.es/Interactive

INDEX TERMS Chemical engineering, control engineering education, cooperative virtual lab, educational simulation, Modelica, object oriented modeling, process system engineering.

I. MOTIVATION AND SIGNIFICANCE

Collaborative virtual labs are effective pedagogical tools in collaborative learning, or group learning, as they allow students to co-operate, combining their efforts to find the solution to a common problem. Provide collaborative features to virtual labs is an active research field. Three different approaches are found in the literature [1]: (1) inserting the virtual lab in a Learning Management System (LMS), (2) embedding the virtual lab into a Virtual World (VW), and (3) supporting multiple participants to handle the same virtual lab by allowing the participants to interact simultaneously on the same virtual lab.

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

Embedding the virtual lab in an LMS or a VW allows its users to employ the collaboration tools, synchronous and/or asynchronous, that the LMS or VW already has to share ideas, information, and results concerning the virtual lab, etc. Besides, users can share information while interacting with the virtual lab if they use synchronous tools such as chats, voice over IP, or videoconferences. LMS typically include chats, videoconferences, whiteboards, forums, wikis, on-line forums, discussion boards, and mailing lists and most VW include instant messaging, 3D motion in the VW, and voice over IP channels. On the other hand, we can share the same experience by allowing several participants to handle the same virtual lab. This is the approach that we have followed. To this end, the virtual lab becomes the main source of communication among participants through real-time and synchronized communication among virtual labs. We have developed a framework to implement collaborative virtual labs which follow this third approach.

Process System Engineering (PSE) is part of the curriculum in engineering studies such as aerospace, mechanical, chemical, industrial and electrical. PSE covers a wide range of topics, such as [2]: system modeling and simulation, optimization, dynamics and control, and process and plant design. To master these topics, it is important not only to have a good theoretical background but also an engineer ability, i.e. insight and intuition, usually obtained by means of many hours of laboratory work, that can be reduced by using virtual labs. Virtual labs have become widely used in distance universities, where students don't have so many in-person practical lessons. There are many examples of virtual labs for PSE education in literature [3]–[5], but there is a lack of frameworks that facilitate the easy implementation of collaborative virtual labs for PSE education based on complex multi-domain models.

Virtual labs are essentially composed of three parts: the simulation of a mathematical model; the interactive student-to-model interface, called the virtual lab view; and a narrative that typically describes the learning outcomes and activities. Interactivity and visualization are interrelated in virtual labs: students are allowed to change model variables by manipulating the view graphic components and can observe the model behavior by means of animated visualizations. Visualization is an important aid to illustrate the complex problems that arise in PSE [6]. Collaborative virtual labs have several instances of the virtual lab view, which are typically executed on different computers, and allow several students cooperatively interact with the same model simulation run.

In this article, a new major release of the Interactive Modelica library is presented. Its most relevant feature is supporting the implementation of collaborative virtual labs, with multiple instances of the same view that can be executed on different computers, facilitating the interaction of several students with the same simulated model. The communication layer of the library has been completely changed to allow efficient synchronization among the simulation model and several views: every view has to reflect the same model behavior at the same time, and interactive changes on the model state are allowed by manipulating any of the views. Additional visualization components are also provided. The code of this new release, named Interactive 3.0, has been developed to be fully compatible with OpenModelica and Dymola, and it has been tested with Dymola 2021 and OpenModelica 1.16 64 bits.

Interactive 3.0, that can be freely downloaded from [7], is distributed as a Modelica library named Interactive, along with two dynamic-link libraries (DLL) named TCPFunctions and InteractiveLib. Interactive 3.0 is geared to Windows systems because some of its files (VTK, TCPFunctions, Qt, and InteractiveLib libraries) are specific for Windows 64 bits operating systems. TCPFunctions uses the windows socket library, so its code should be changed to be ported to Linux. VTK, Qt, and InteractiveLib can easily be recompiled to a Linux version. There exist Linux versions of the most popular Modelica environments such as Dymola and OpenModelica, that can be used to simulate any Modelica model.

The main contribution of this paper is to provide a free framework for developing collaborative virtual labs using only Modelica. To this end, a new synchronization algorithm and a new communication layer have been developed and included in the Interactive Modelica library. The communication architecture has a fundamental role in these virtual labs, and will be explained in Section V. Additionally, the Interactive Modelica library has been modified to be compatible with the Open-source Modelica environment OpenModelica, facilitating a free solution to collaborative virtual lab implementation.

The structure of the paper is as follows. Firstly, the related work is discussed in Section II, and the design principles and implementation of the Interactive 3.0 Modelica library are discussed in Sections III to V. The software architecture of the Interactive 3.0 Modelica library, focusing on the classes that include the communication code and the TCPFunctions DLL, is discussed in Section III, and the InteractiveLib DLL is described in Section IV. The most relevant aspects of the communication framework are discussed in Section V. Finally, the Interactive 3.0 Modelica library use is illustrated in Section VI through the development of a collaborative virtual lab based on the Tennessee Eastman simplified model [8], [9], a well-known process in chemical engineering. This virtual lab is used to get insight into the behavior of this chemical process plant, and to apply different multi-loop control and optimization strategies.

II. RELATED WORK

The object-oriented modeling language Modelica [10] greatly facilitates the description of hybrid dynamic models, non-causal models described by systems of differential-algebraic equations (DAE) and events. Modelica provides language constructs to describe time and state events, to reinitialize state variables, to update discrete-time variables, to declare object-oriented constructs, connectors to specify the interaction between models, etc. Besides, there has been an international effort to provide Modelica libraries in different domains (hydraulic, thermal, chemical, mechanical, etc.), some of them free, well documented, and ready to be used. As this type of mathematical model (i.e, hybrid-DAE system) is widely used in process modeling, Modelica is well suited for implementing the type of models found in PSE and process industry.

The Modelica modeling environment (e.g., Dymola [11] and OpenModelica [12], [13]) makes the required manipulation on the model (e.g., remove redundant equations, analyze the computational causality, sort the equations, DAE index reduction, symbolical manipulation of the linear systems of simultaneous equations, tearing of nonlinear systems of simultaneous equations), and generates the executable code adding numeric solvers. These environments usually have graphical model editors that allow composing the model by simply dragging and dropping the components of the Modelica model libraries.

Different research lines have been followed to facilitate interactive simulation and visualization of Modelica models. One approach is to provide Modelica modeling environments with capabilities for interactive simulation. The OpenModelica Connection Editor (OMEdit) provides an interface to the interactive simulation module (OMI) in order to support interactive changes in the model parameters during the simulation run [14]. A web service communication layer for OpenModelica [12], [13] was implemented, and employed in [15], [16] to create interactive online simulations that allow users to change model parameters during the simulation run.

Other approaches are based on cosimulation. Virtual labs for control education were developed in [17] combining Dymola and Ejs [18]; and Dymola and Sysquake [19]. Cosimulation and model exchange based on Functional Mock-up Interface [20] is exploited in [21]–[23] to develop interactive simulations of Modelica models.

The Modelica_DeviceDrivers library [24] allows setting the value of model input variables using external devices (e.g., keyboard, joystick, etc.). The MultiBody Modelica library includes animated objects to visualize the simulation results. Two Modelica libraries for visualization are Modelica3D [25] and Visualisation [26].

VirtualLabBuilder [17] and Interactive [27] are two free Modelica libraries that facilitate composing the virtual lab view; establishing the relationship between model variables and the visual properties of the view; and linking the HTML pages that constitute the virtual lab narrative. The virtual lab view is described instantiating and connecting the graphic elements provided in VirtualLabBuilder or Interactive, forming a hierarchical tree that reflects the virtual lab view layout. VirtualLabBuilder and Interactive graphic elements (e.g., containers, animated 2D geometric shapes, basic elements and interactive controls) are Java and C++ code generators, respectively. During the initialization stage of the virtual lab simulation, the virtual lab view application is automatically generated, and the bidirectional model-view communication is established. This is accomplished by the Modelica classes describing the graphic elements, which contain in their initialization sections calls to functions aimed to write this code. The virtual lab view generated by VirtualLabBuilder is programmed in Java and doesn't contain 3D geometric shapes, whereas the view generated by Interactive is programmed in C++ using the Qt, VTK, and Qwt libraries, has better graphic quality and includes 3D geometric components.

For a Modelica model to be employed in a virtual lab implemented using VirtualLabBuilder or Interactive, it needs to be adapted according to the methodology proposed in [28]. All model quantities that will be allowed to change interactively (the so-called interactive quantities) have to be selected as state variables. In particular, model parameters are transformed into interactive quantities by describing them as state variables with zero time-derivative. As different interactive actions may require different selections of the state variables, this approach may require executing in parallel several simulation instances, with different selections of the state variables. Modelica facilitates model developers to select the model state variables, and supports the reinitialization of state variables at events.

Previous versions of the VirtualLabBuilder and Interactive Modelica libraries facilitate the development of single-user virtual labs whose model and view run locally, on the same computer. VirtualLabBuilder is only compatible with Dymola, whereas the latest version of Interactive can be used in combination with other Modelica modeling environments. Interactive 3.0 has been tested with Dymola and OpenModelica in Windows.

III. THE INTERACTIVE MODELICA LIBRARY

The Interactive 3.0 Modelica library is structured into four packages (see Fig. 1). The VLabModels, ViewElements and Examples packages contain the Modelica classes that the virtual lab developer employs. The src package contains partial classes and Modelica functions not intended to be directly used by virtual lab developers.

The src.CServer package encapsulates the C functions included in the TCPFunctions DLL. The TCPfunction DLL includes functions written in C to create a server, to attend requests from clients, and to send and receive TCP messages. There are calls to these functions from the following three partial classes of the Interactive Modelica library: PartialView, Drawable and SendElement.

To perform these communication tasks, the TCPFunctions DLL includes the following C functions:

- startNClientCserver: starts the server and waits until a determined number of views have been connected. The number of views is a parameter of the VirtualLab class. This function returns a vector with the socket number of each connected view, which is necessary to send/receive data to/from these views.
- sendOutput: sends a string as a TCP message to a view. The string contains the value of the model variables that are visualized by the views.
- getVarValues: receives a TCP message containing a string with the following information: number of changes performed on the view, a reference to the changed model variables, and their new value.
- sendChalk: sends a 1/0 value depending on whether the changes performed on the view have been applied or not to the model.

The VLabModels package includes the PartialView and VirtualLab classes. The virtual lab is described as a Modelica class that includes an object of the VirtualLab class, which has two objects: Model and View. The classes of these two objects, initialized to a null class, must be redeclared to the classes describing the physical model and the view, making use of the Modelica facility to redeclare the class of an object [29]. The class describing the view must inherit from

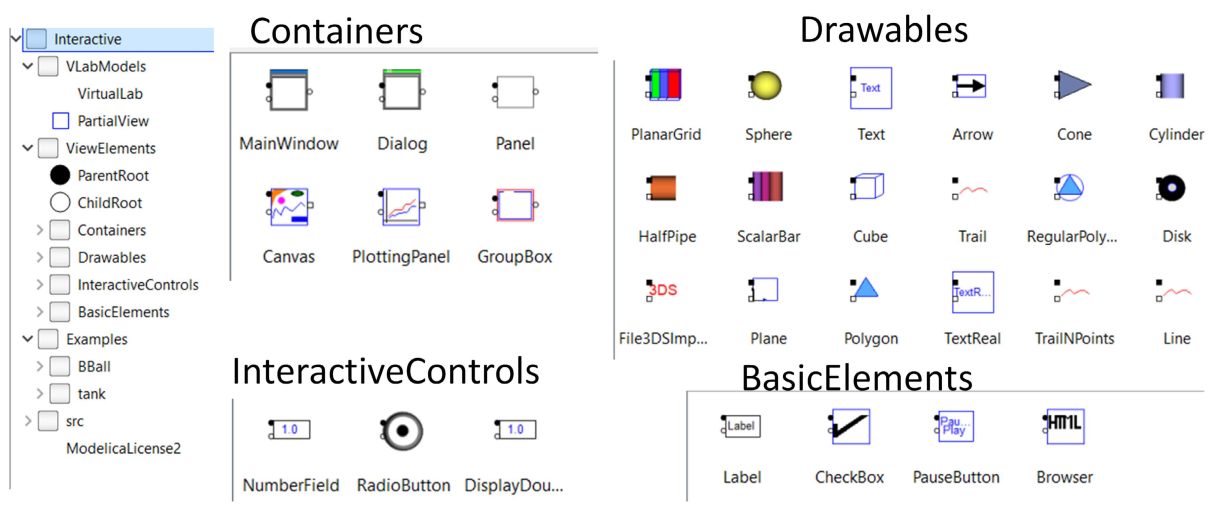


FIGURE 1. **The interactive 3.0 Modelica library.**

the PartialView abstract class. The procedure to build the virtual lab will be illustrated in Section VI by means of a case study.

The PartialView class has been redesigned to include the following code concerning the communication:

- Declaration of parameters and global variables related to the server-clients communication, such as the array of socket descriptors.
- Initial algorithm section, executed in the Modelica environment before the simulation starts, includes a call to the function startNClientCserver.
- When clause, whose code is executed at regular steps. The code of this clause includes a call to the getVarValues function, and a sentence changing the value of a global boolean variable named *refreshView*. The change of the value of the *refreshView* variable, triggers an event in the SendElement and Drawable partial classes that causes the execution of their communication code. We call the objects that inherit these two partial classes interactive objects, and the number of interactive objects existing in the view description *nI*.

The Container package includes Modelica classes describing windows, panels, the plot container (PlottingPanel class) and the animation container (Canvas class). These classes don't include any communication code.

The Drawables package includes Modelica classes describing components that are hosted inside a PlottingPanel model or a Canvas model. It can be selected by the developer whether the variables of the objects of these classes (such as radius, position, etc.) send or not their value each communication interval to the view (i.e., are or not interactive). The graphic components included in these packages inherit from the Drawable class, which has been modified to include the code to send the interactive variable values to every view. The Drawable class is a partial class that has two global variables: an array with the socket descriptors corresponding to the views and the *refreshView* variable. This class includes a when clause that is executed only when there is a change of value in the *refreshView* variable. This *when* clause includes calls to the sendOutput function to send the interactive data associated to the graphic component to every view. When it is detected that every TCP connection is down, the simulation is terminated.

The InteractiveControls package includes Modelica classes describing components such as numeric boxes that are hosted inside a window or panel. These components inherit from the ControlElement class, which includes code to change the variable value associated with the component. This variable is linked to a model variable, allowing to change the value of the model variables in a way transparent to the user. Some of these components inherit additionally from the SendElement class, which has the same communication code that was included in the Drawable class.

The BasicElements package contains four classes: Label, CheckBox, PauseButton, and Browser. Objects of these classes can be included inside a window or panel. PauseButton creates a button for pausing and resuming the simulation. Browser creates a container of documentation in HTML format.

IV. THE InteractiveLib DLL

The InteractiveLib DLL contains the C++ classes of the view graphic elements and the code to communicate with a server. The C++ source code generated from the Modelica view description includes instantiations of InteractiveLib DLL classes. The InteractiveLib DLL has been programmed in C++ using Qt 5.12, Qwt 6.1, and VTK 7.1. libraries.

Qt [30] is an object-oriented cross-platform framework aimed to develop applications that use C++ as a native language and is available under the terms of GNU Lesser General Public License. It was originally conceived to facilitate the development of graphical user interfaces (GUIs) using its Widgets module, but nowadays provides modules for networking, databases, OpenGL, etc., and bindings for different programming languages. A main feature of Qt is its mechanism for communication between objects, the signal slot mechanism. We have employed its capabilities for networking, OpenGL, and the signal and slot mechanism in the InteractiveLib implementation. The Qt library has been used to develop the C++ code of the view corresponding to the communication between the view and the model, the containers and the interactive elements (i.e., sliders, check-boxes, etc).

Qwt [31] is a set of widgets for technical applications written in C++ and freely distributed as a set of files that must be compiled and installed on the target system. Some of its plots and trails are included in the InteractiveLib DLL.

VTK [32], [33] is an open-source toolkit for creating leading-edge visualization and graphics applications that manipulate and display scientific data, licensed under the BSD license. Its core functionality is written in C++, and runs on Linux, Windows, and Mac. VTK provides a rendering abstraction layer over the underlying graphics library (OpenGL for the most part), and tools for 3D rendering, modeling, image processing, a suite of widgets for 3D interaction, volume rendering, and extensive 2D plotting capability. It supports a wide variety of visualization algorithms and advanced modeling techniques, and it takes advantage of both threaded and distributed memory parallel processing for speed and scalability, respectively. There is a special class included in VTK, named QVTKWidget, to display a VTK window in a Qt window. VTK is used in InteractiveLib DLL, in combination with Qt (by using the QVTWidget class), to create the 3D animation elements such as spheres, Halfpipe, ScalarBar, etc. and for the rendering and the visualization.

The view code has two threads: a thread to handle the graphical user interface and a thread exclusively dedicated to communicate with the simulation model.

The communication thread connects to the server, sends the new model variable values that have been modified due to the user action on the interactive controls (e.g., sliders), gets a message from the server informing whether or not the new values have been modified in the model, and obtains the model variables values needed to refresh the view (see Fig. 3).

The graphic components included in the Container, Drawables, InteractiveControls and BasicElement packages of the Interactive 3.0 Modelica library have an analogous class in the InteractiveLib DLL. There is a `MainWindow` class in Interactive, and a MainWindow class in InteractiveLib implemented using Qt. This MainWindow class is in every view description and includes the code to render the animation and the graphs; and to close the view application. The classes hosted inside a `PlottingPanel` and a `Canvas` model have corresponding classes in the InteractiveLib DLL, whose source code has been developed using Qwt and VTK respectively. There is a special class included in VTK, named QVTKWidget, to display a VTK window in a Qt window.

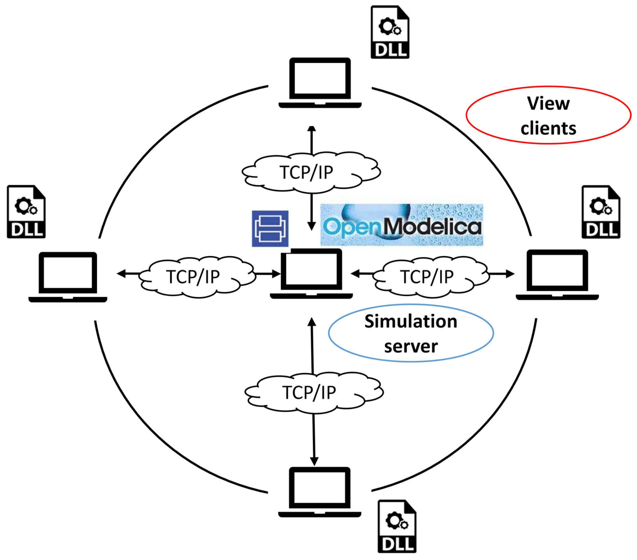


FIGURE 2. The centralized network of the collaborative system.

The corresponding class of the Interactive 3.0 `Canvas` class inherits from QVTKWidget.

V. COMMUNICATION FRAMEWORK

The communication framework is based on the TCP protocol and a multiple client server architecture. There is one model simulation and multiple views, that are connected to the model simulation using a peer to peer centralized network, as is shown in Fig. 2. The communication engine is embedded in each view and the model, which are always synchronized in the same model state. The model simulation stops at regular time steps to exchange TCP messages with each view, a process explained below.

A. SERVER: MODEL SIMULATION

The virtual lab model is a Modelica class that has three parameters to set up the model-views communication: the port number where the views will be connected, number of views to be connected with the simulation model (nV), and time stamp between two successive model-view communications (communication interval).

The Modelica class describing the virtual lab model includes an object describing the Modelica model and an object describing the view, and equations connecting the model and the view variables. The procedure to build the virtual lab will be illustrated in Section VI by means of a case study.

Once the virtual lab model is executed, the model simulation starts a TCP server that attends new TCP requests from views in a fixed port, storing a connection handler for each view connection in a global variable of the type array declared in the `PartialView` class. The server waits until the predefined number of views are connected to it, and then the simulation begins.

The simulation is stopped at regular steps defined by the communication interval using the sample built-in Modelica

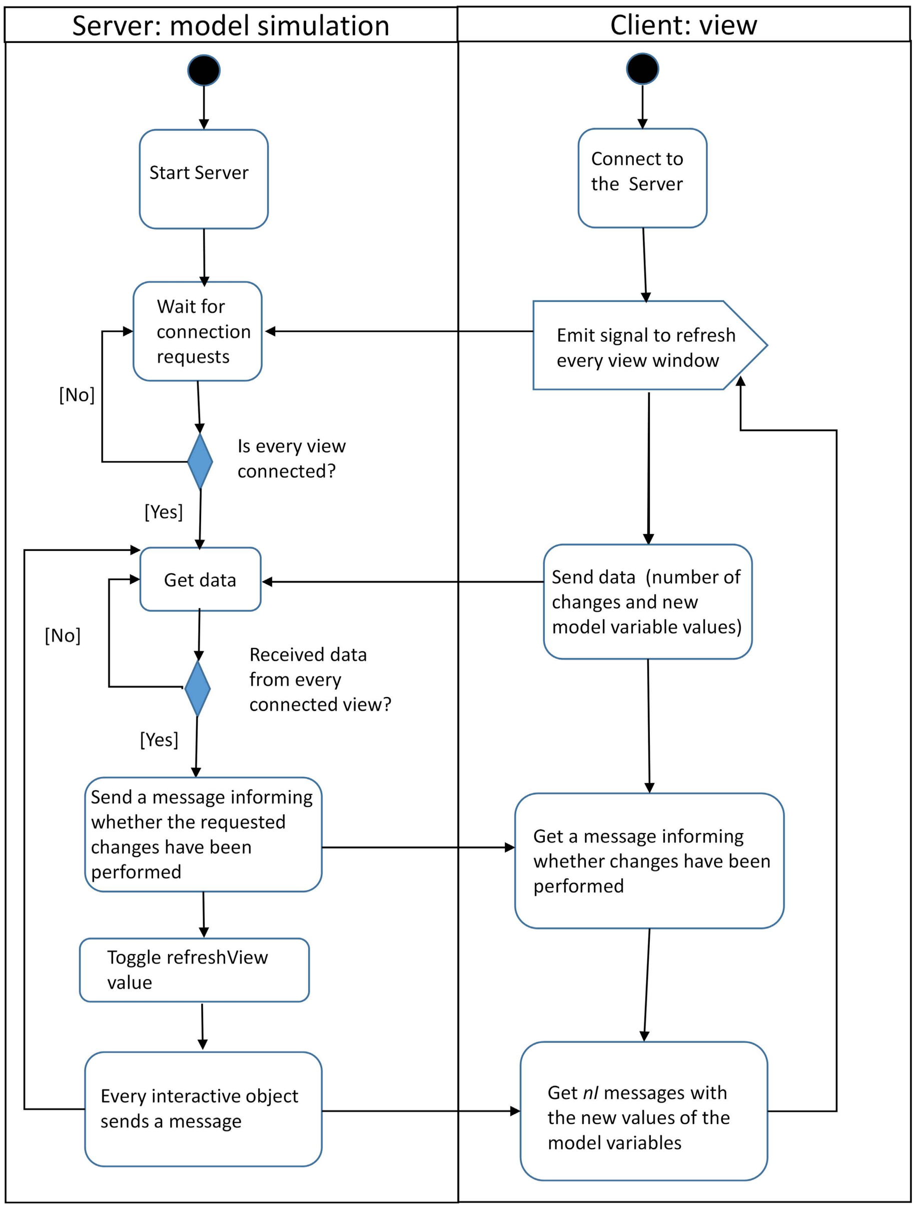

Server: model simulation
Client: view
Start Server
Wait for connection requests
[No]
Is every view connected?
[Yes]
Get data
[No]
Received data from every connected view?
[Yes]
Send a message informing whether the requested changes have been performed
Toggle refreshView value
Every interactive object sends a message
Connect to the Server
Emit signal to refresh every view window
Send data (number of changes and new model variable values)
Get a message informing whether changes have been performed
Get nI messages with the new values of the model variables

FIGURE 3. UML activity diagram describing the exchange of information between the server and one of the clients.

function. At these time instants, there is a synchronized and bidirectional flow of messages between the model server and each view client.

At these events, the following actions take place sequentially (see Fig. 3).

1) The server waits for each view to send the number of changes performed by the user and the new values of the model variables (nV messages). The server performs or not the changes from a determined view.
2) The server sends a message to each view informing whether the requested changes have been made, and waits until every client acknowledges the message reception (nV messages).
3) The value of the boolean variable *refreshView* is changed.
4) This change triggers an event that causes every interactive object (i.e., objects whose superclass is Drawable or SendView) to send a message to each view client. Thus, nI messages are sent to the nV clients ($nI \cdot nV$ messages).

When the server detects that every client has been disconnected, the simulation is terminated. If there are simultaneously several views with a number of changes greater than zero, the model has to select one of these views and executes only the changes performed by manipulating this view. The selection of this view can be implemented in different ways. We have designed a simple selection procedure to reduce the time to perform the selection and the number of transmitted messages.

The selection procedure is as follows. During the connection of the view to the server, a priority is assigned to each view, depending on the time instant that the view asked the server to be a client. The first view is assigned the highest priority, and the last view the lowest priority. When two or more views send a number of changes different from zero, the selected view is the one of them with the highest priority. The model implements only the changes of this selected view. As the selection procedure is known beforehand, the instructor can use this information to prioritize a view with respect to the rest.

The total number of messages in each communication instant depends linearly on the number of views and the number of interactive objects in each view. Thus, the time involved in the communication increases linearly with the number of views and the complexity of the view.

B. CLIENTS: VIEW

The virtual lab view is a C++ application that contains objects of classes included in the InteractiveLib DLL, that are described in Section IV. This application has an object called CommThread, a thread that handles the connection to the model simulation through the TCP channel. This object includes an array that contains pointers to each view object whose properties are set to new values sent from the model simulation (i.e., interactive object). Thus, the size of this array is *nI*.

The CommThread object connects to the server, and then starts a loop that repeats the following steps until the view is closed.

1) It emits a signal to refresh every view window.
2) Then, sends the number of changes performed by the user and the new model variable values that have been modified due to the user action on the interactive controls (e.g., sliders).
3) It gets a message from the server informing whether or not the new values have been modified in the model.
4) It gets *nI* messages from the server, one message from each interactive object with the new model variables values needed to refresh the view (see Fig. 3). The message includes a number to identify the interactive object, which is required to obtain the pointer to the corresponding object and update the values accordingly.

VI. TENNESSEE EASTMAN SIMPLIFIED PROCESS VIRTUAL LAB

The Tennessee Eastman Process model [34] describes a real chemical process that contains a separator/reactor/recycle arrangement, involving two simultaneous gas-liquid exothermic reactions. This non-linear dynamic model has been employed as a benchmark for manufacturing process control, statistical process monitoring, sensor fault detection, and

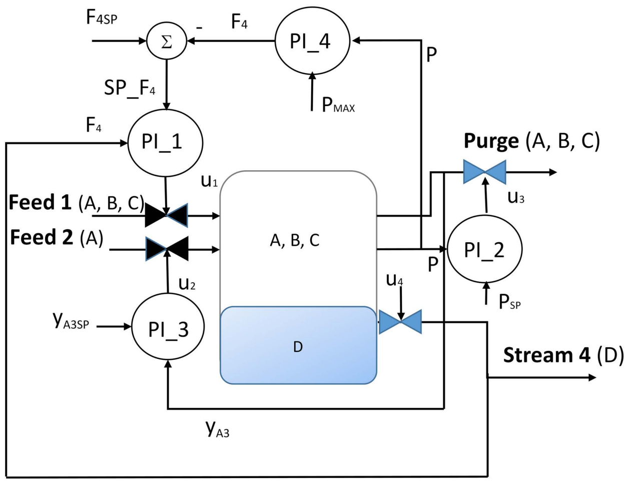


FIGURE 4. Diagram of the plant with the four PI controllers.

identification of data-driven network models. The Tennessee Eastman Simplified Process (TES) model [8] is a simplification of the Tennessee Eastman Process model. It considers only one process unit, consisting of a combination of a reactor and a separator.

The process unit of the TES model has 2 input flows (named Feed 1 and Feed 2) and two output flows (named Purge and Stream 4). Feed 1 contains the non-condensable gases A and C, and trace amounts of an inert gas B. Feed 2 only contains component A. The irreversible reaction $A + C \longrightarrow D$ occurs in the vapor phase under isothermal operating conditions. The product D is a non-volatile liquid. The process unit contains a vapor phase, composed of the A, B, and C ideal gases, and a liquid phase composed of pure D. Purge is a gas mixture flow composed of the A, B, and C ideal gases. Stream 4 contains only the liquid D.

A. EDUCATIONAL GOALS

A collaborative virtual lab is designed to teach students the dynamic behavior of the TES model, how to operate, control and optimize this process unit, and the effect of disturbances. The TES model is an example of a multi-input multi-output, nonlinear, open-loop unstable system with fast and slow dynamics. The control challenge is to maintain a specified product rate by manipulating the Feed 1, Feed 2, and Purge flows.

The multi-loop control strategy proposed in [8] is implemented. A diagram of the controlled plant is shown in Fig. 4. It consists of four PI controllers, PI_1 to PI_4, whose pairs of controlled-manipulated variables are respectively: the production rate (F_4) and the valve position for Feed 1 (u_1); the reactor pressure (P) and the valve position for Purge (u_3); the concentration of component A in Purge (Y_{A3}) and the valve position for Feed 2 (u_2); and the reactor maximum pressure (P_{MAX}) and the correction to the production rate setpoint (F_{4SP}). The operating pressure must be kept below the shutdown limit of 3000 kPa. Students are asked to solve

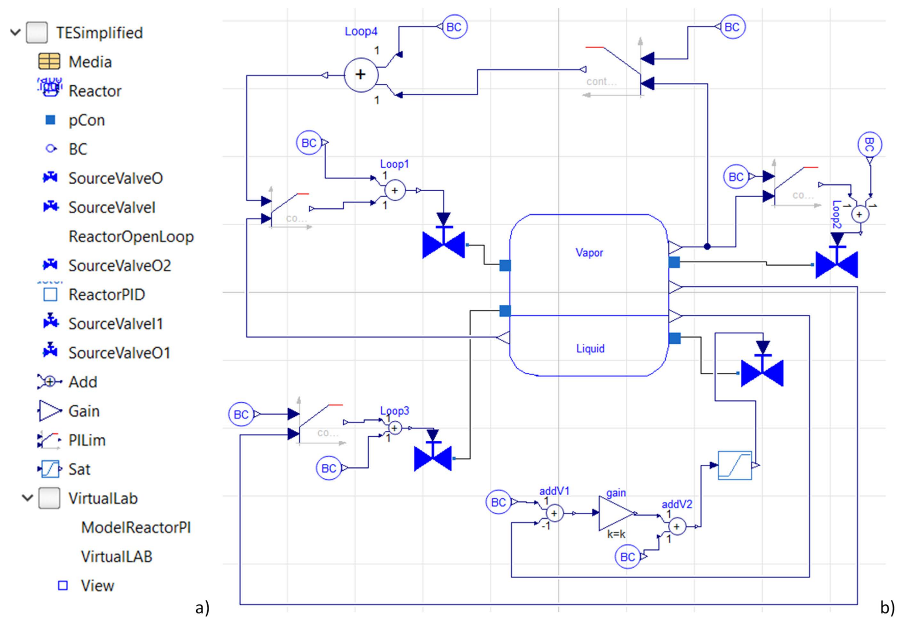


FIGURE 5. The TESimplified Modelica library: a) structure; and b) diagram of the ReactorPID model.

the tasks working in a group, using the collaborative virtual lab.

B. VIRTUAL LAB MODEL

A Modelica model of the TES process [8] was developed in [9]. The methodology proposed in [28] has been applied: the interactive quantities have been selected as state variables using the Modelica facilities to set the state variables, and the interactive parameters have been redefined as state variables with zero time-derivative. The virtual lab view is composed using Interactive 3.0. The complete virtual lab is distributed in a library named TESimplified, written in Modelica 3.3 and tested using Dymola 2021, and OpenModelica 1.16 64 bits under Windows 2010.

The structure of the TESimplified Modelica library is shown in Fig. 5a. The TES process unit is described in the Reactor model. The PI controllers have limited output, anti-windup compensation and setpoint weighting [35]. The diagram of the controlled plant, described in the ReactorPID model, is shown in Fig. 5b.

The parameters of the four PI controllers (i.e., proportional gain, integral time constant, anti-windup compensation time constant, setpoint weight, and upper and lower limits) are interactive quantities of the virtual lab. The values of these parameters given in [8] are taken as initial values for these interactive quantities.

Other interactive quantities of the virtual lab are the setpoints of the four PI controllers, the composition of Feed 1, and the parameters of the reaction rate equation. The reaction rate (R_D) is assumed to depend only on the partial pressures of A (P_A) and C (P_C) as follows: $R_D = k_0 \cdot P_A^{\alpha} \cdot P_C^{\beta}$. The values of k_0, α and β given in [8] are taken as initial values for these interactive quantities: $k_0 = 0.00117$, $\alpha = 0.5$, $\beta = 0.4$, with R_D expressed in kmol/h, and P_A and P_C in kPa.

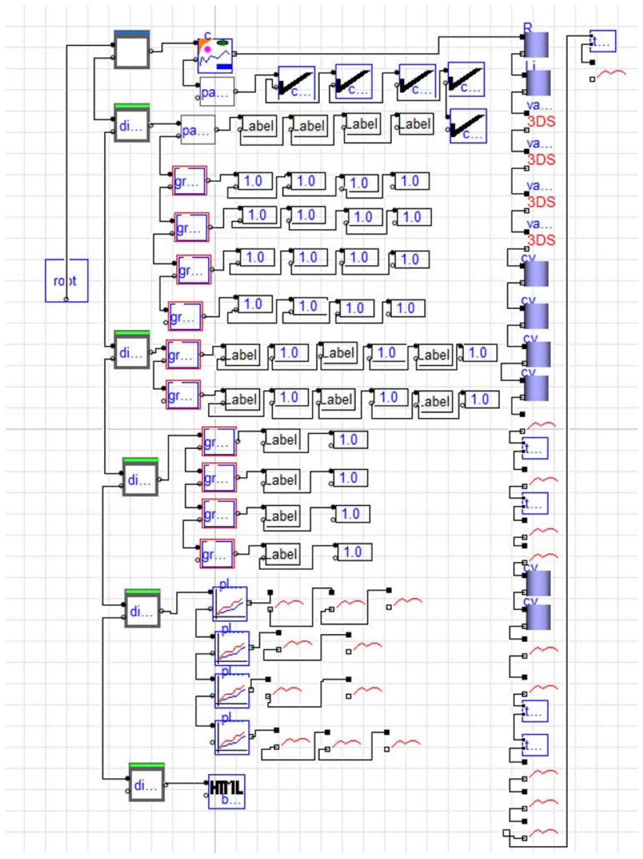
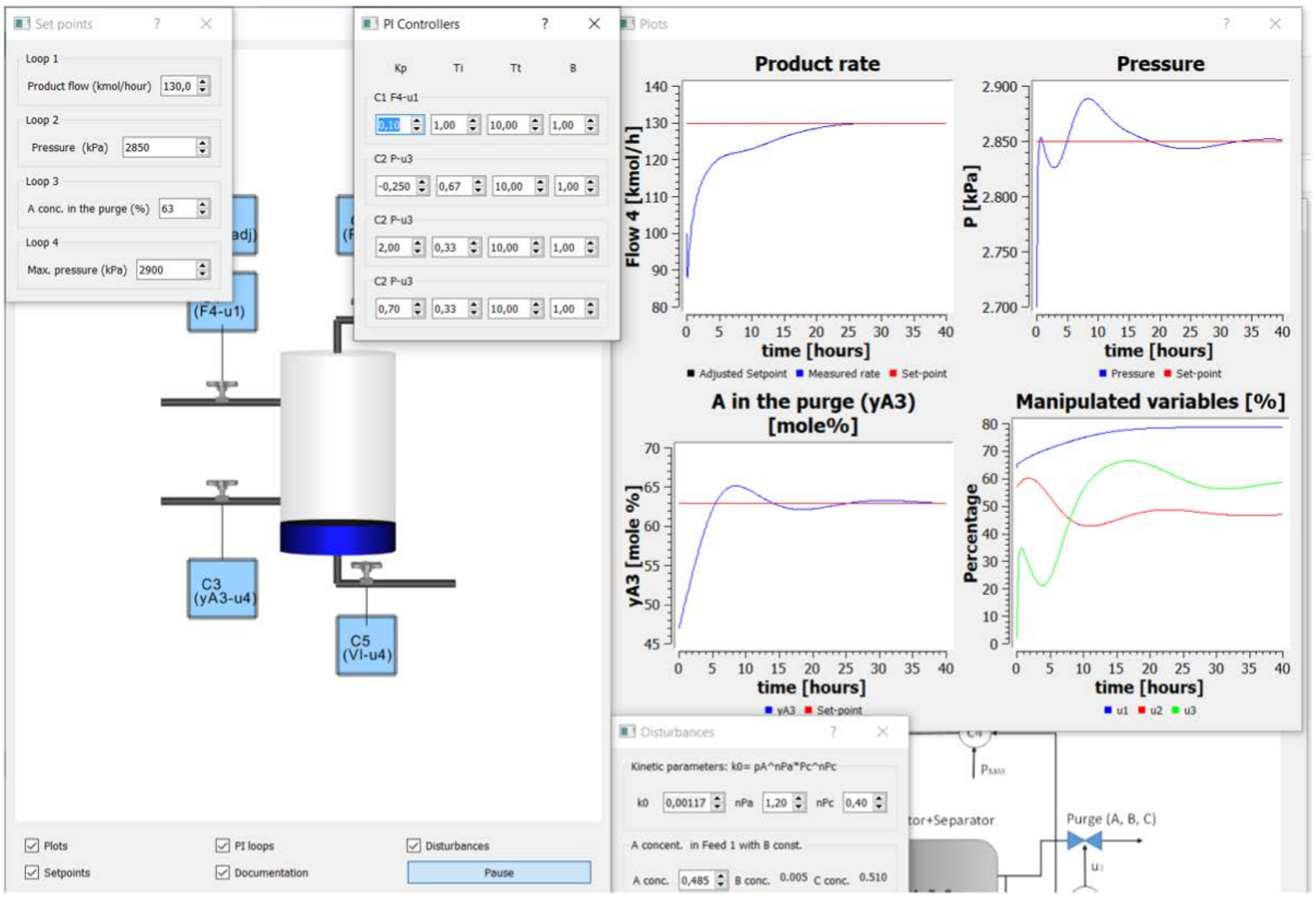


FIGURE 6. Modelica model that describes the virtual lab view.


Set points

Loop 1
Product flow (kmol/hour) 130,0

Loop 2
Pressure (kPa) 2850

Loop 3
A conc. in the purge (%) 63

Loop 4
Max. pressure (kPa) 2900

PI Controllers

	Kp	Ti	Tt	B
C1 F4-u1	0,10	1,00	10,00	1,00
C2 P-u3	-0,250	0,67	10,00	1,00
C2 P-u3	2,00	0,33	10,00	1,00
C2 P-u3	0,70	0,33	10,00	1,00

Disturbances

Kinetic parameters: k0= pA^nPa*Pc^nPc
k0 0,00117 nPa 1,20 nPc 0,40

A concent. in Feed 1 with B const.
A conc. 0,485 B conc. 0.005 C conc. 0.510

☑ Plots ☑ PI loops ☑ Disturbances
☑ Setpoints ☑ Documentation

FIGURE 7. Virtual lab view automatically generated from the Modelica description.

C. VIRTUAL LAB VIEW

The virtual lab view is composed graphically by instantiating and connecting elements of the Interactive 3.0 Modelica library. The diagram of the Modelica model that describes the virtual lab view is shown in Fig. 6. The virtual lab view automatically generated from this description is shown in Fig. 7.

The Modelica class that describes the view must, on the one hand, be a subclass of the PartialView class, which is included in the Interactive library and contains the code of the model-view bidirectional communication. On the other hand, it has to contain the components employed to define the view, connected forming a tree structure. The PartialView class contains an object named root. This object must be connected to the rest of the view components following the library connection rules. Thus, this component is the root of the tree structure describing the view (see Fig. 6). As shown in Fig. 6, six components are directly connected with root: the mainWindow component of the MainFrame class, which generates the window shown in Fig. 7, and five components of the Dialog class.

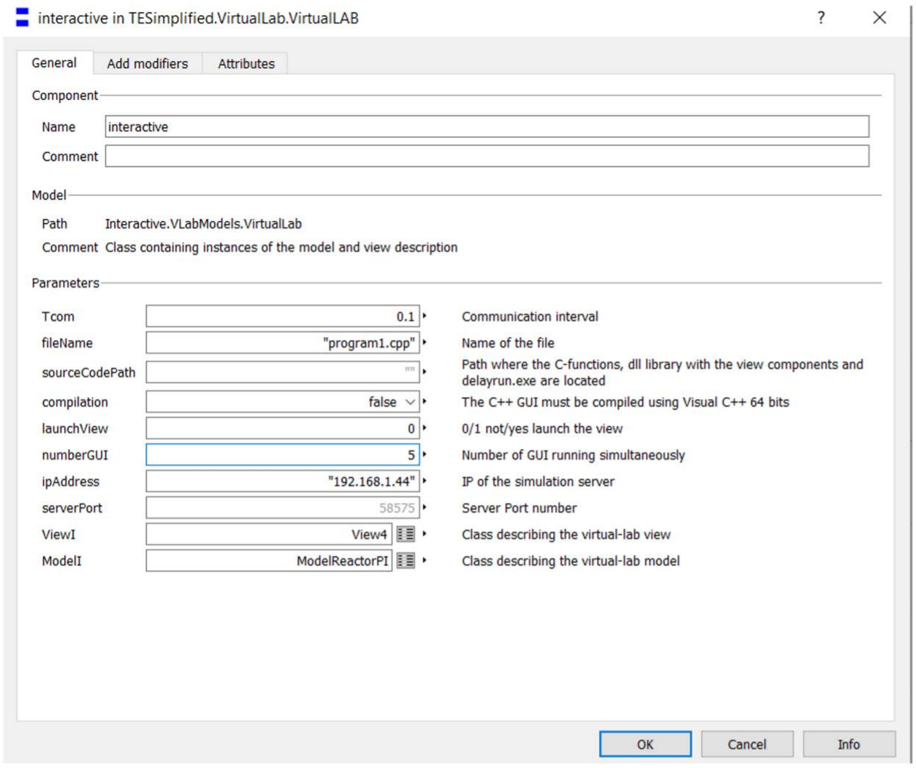
Two containers are placed inside mainWindow: a component of the Canvas class, placed in the center of mainWindow, that contains the 3D animated diagram of the TES model; and a component of the Panel class that hosts interactive controls to pause and resume the simulation, and check-boxes to show and hide dialog windows.

The 3D animated diagram of the TES model is composed of drawable elements. The reactor is represented by components of the Cylinder class. The valves are represented by components of the File3DsImporter class, which imports 3D studio file into the view. The controllers are represented by components of the Text and Line classes.

The interactive controls that allow pause/resume the simulation and show/hide windows are described by components of the PauseButton and CheckBox classes. The virtual lab view contains six dialog windows that allow tuning the PI controllers; to change the composition of Feed 1, the reaction rate parameters and the setpoints of the PI controllers; to display the time evolution of the Stream 4 flow rate, the reactor pressure, the concentration of component A in the purge, and the valve positions; and to show the HTML pages that constitute the virtual lab narrative.

D. VIRTUAL LAB SET UP

The Modelica model that describes the complete virtual lab must instantiate the VirtualLab class of the Interactive 3.0 Modelica library, the Modelica class describing the virtual


interactive in TESimplified.VirtualLab.VirtualLAB

General | Add modifiers | Attributes

Component

Name	interactive
Comment	

Model

Path	Interactive.VLabModels.VirtualLab
Comment	Class containing instances of the model and view description

Parameters

Tcom	0.1	Communication interval
fileName	"program1.cpp"	Name of the file
sourceCodePath	""	Path where the C-functions, dll library with the view components and delayrun.exe are located
compilation	false	The C++ GUI must be compiled using Visual C++ 64 bits
launchView	0	0/1 not/yes launch the view
numberGUI	5	Number of GUI running simultaneously
ipAddress	"192.168.1.44"	IP of the simulation server
serverPort	58575	Server Port number
ViewI	View4	Class describing the virtual-lab view
ModelI	ModelReactorPI	Class describing the virtual-lab model

FIGURE 8. Parameters of the virtual lab Modelica model.

lab model, and the Modelica class describing the virtual lab view. In addition, this model describing the complete virtual lab has to contain the equations equaling the view variables to the corresponding model variables. Finally, values have to be assigned to the following parameters of the VirtualLab class: the length of the model-to-view communication interval, the names of the Modelica classes that describe the model and the view, the number of views, and the IP address of the computer where the simulation model is running (see Fig. 8).

E. TRANSLATION TO EXECUTABLE CODE AND LAUNCH

The Modelica description of the complete virtual lab is translated into executable code using a Modelica modeling environment. The Interactive 3.0 Modelica library has been tested using Dymola 2021 and OpenModelica 1.16. Then, the generated executable code is launched in the server computer, starting the simulation run. At the initialization stage of the simulation, the C++ code of the view application is automatically generated in the server computer, and the simulation run waits for the clients (i.e., the virtual lab view applications) to connect.

This C++ code of the view application, which has been automatically generated in the server computer, needs to be copied to the students' computers and compiled. The InteractiveLib, Qt, VTK, and Qwt DLL also need to be copied to the students' computers. Next, the compiled copies of the view application are launched in the students' computers. When the view application is launched in a student's computer, this application gets connected to the server, and the model-view communication is automatically established.

Once the specified number of views are connected (this number is a VirtualLab class parameter), the simulation initialization is completed in the server and the interactive simulation proceeds. The code that needs to be copied to the students' computers doesn't change between successive runs of the virtual lab, so view installation only needs to be done once. Neither Dymola nor OpenModelica needs to be installed on the students' computers.

F. VIRTUAL LAB USE

The virtual lab narrative is presented to the students through HTML pages linked to the virtual lab view. This narrative describes the virtual lab pedagogical goals, the TES process and its multi-loop PI control system, and how to use the virtual lab. Students are asked to read the narrative and to experiment in groups with the collaborative virtual lab to complete the proposed activities. One of these collaborative activities consists in dividing the students into four groups, and assign to each group the tuning of one of the four PI controllers. Next, students are asked to describe and explain the observed influence of the other three controllers' tuning on their own.

Some other collaborative activities proposed to the students are listed below. Some of them are the scenarios for process control discussed in [8].

1) Set to zero the value of the PI proportional constant, which is equivalent to cancel the control action. Observe the unstable dynamic of the process: the reactor pressure increases until the shutdown limit is met.
2) Observe the controlled system and describe its evolution. How long does it take to stabilize the system?
3) Change the control parameters. Analyze the limits for controlling the plant.
4) Without touching other controls, comment on what happens after modifying the product flow setpoint to 140 kmol/h.
5) Without touching other controls, comment on what happens after reducing the maximum pressure in the reactor to 2700 kPa. Is it possible to reach the following product flow setpoint: 130 kmol/h?
6) Analyze the regulation during a disturbance in the feed composition. Keep the production rate, while the molar fraction of component A in Feed 1 changes step-wise from the nominal 0.485 to 0.45.
7) Analyze the effect of a drift in the kinetic parameters. Change the proportionality constant k_0 of the reaction rate equation (i.e., $R_D = k_0 \cdot P_A^{\alpha} \cdot P_C^{\beta}$) from 0.0017 to 0.001, while the exponent β drifts from 0.4 to 0.35.

VII. CONCLUSION

A new major release of the Interactive Modelica library has been presented. Its most relevant feature is to facilitate the implementation of collaborative virtual labs based on Modelica models, using only the Modelica language. Collaborative virtual labs are composed of several synchronized views that interact with a common model simulation run.

The main challenge has been to design an efficient communication between the model and views that is transparent to virtual lab developers. To this end, we analyzed the minimum information needed to be transmitted, developed a synchronization algorithm, and redesigned the Modelica classes containing this code exploiting the advantages of Modelica object orientation and creating partial classes so that they could be easily extended to create new library components.

The use of the Interactive 3.0 Modelica library has been illustrated by discussing the implementation of a collaborative virtual lab for control education, based on the Tennessee Eastman Simplified Process. The Interactive library facilitates easy definition of the virtual lab view and the model-view connection. The obtained virtual lab has good graphic quality and performance, and is a collaborative learning tool that allows students to work in groups for achieving their learning goals.

The Interactive 3.0 Modelica library is freely distributed under Modelica License 2. The C++ code automatically generated by the components of the Interactive Modelica library uses the Qt, Qwt, and VTK libraries. This library is compatible with OpenModelica and Dymola, having been tested with Dymola 2021 and OpenModelica 1.16 64 bits. As OpenModelica is an open-source tool, this compatibility implies that we have a completely free framework to develop virtual labs.

REFERENCES

[1] R. Heradio, L. Torre, D. Galan, F. Cabrerizo, E. Herrera-Viedma, and S. Dormido, "Virtual and remote labs in education: A bibliometric analysis," *Comput. Educ.*, vol. 98, pp. 312–313, Jul. 2016.

[2] I. T. Cameron, S. Engell, C. Georgakis, N. Asprion, D. Bonvin, F. Gao, D. I. Gerogiorgis, I. E. Grossmann, S. Macchietto, H. A. Preisig, and B. R. Young, "Education in process systems engineering: Why it matters more than ever and how it can be structured," *Comput. Chem. Eng.*, vol. 126, pp. 102–112, Jul. 2019.

[3] M. Johansson, M. Gafvert, and K. J. Astrom, "Interactive tools for education in automatic control," *IEEE Control Syst.*, vol. 18, no. 3, pp. 33–40, Jun. 1998.

[4] L. Marin, H. Vargas, R. Heradio, L. Torre, J. Diaz, and S. Dormido, "Evidence-based control engineering education: Evaluating the LCSD simulation tool," *IEEE Access*, vol. 8, pp. 170183–170194, 2020.

[5] J. Diaz, R. Costa-Castello, and S. Dormido, "An interactive software tool to learn/teach robust closed-loop shaping control systems design," *IEEE Access*, vol. 9, pp. 125805–125819, 2021.

[6] M. A. Rau, W. Keesler, Y. Zhang, and S. Wu, "Design tradeoffs of interactive visualization tools for educational technologies," *IEEE Trans. Learn. Technol.*, vol. 13, no. 2, pp. 326–339, Apr. 2020.

[7] A. Urquia, C. Martin-Villalba, M. A. Rubio, and V. Sanz. (2022). *Some Free Modelling & Simulation Resources*. Accessed: May 17, 2022. [Online]. Available: http://www.euclides.dia.uned.es/

[8] N. L. Ricker, "Model predictive control of a continuous, nonlinear, two-phase reactor," *J. Process Control*, vol. 3, no. 2, pp. 109–123, May 1993.

[9] C. Martin-Villalba, A. Urquia, and G. Shao, "Implementations of the Tennessee Eastman process in Modelica," in *Proc. 9th Vienna Int. Conf. Math. Modeling (MATHMOD)*, 2018, pp. 619–624.

[10] Modelica Association. (2022). *Modelica Association Website*. Accessed: May 17, 2022. [Online]. Available: https://www.modelica.org

[11] (2022). *Dymola*. Accessed: May 17, 2022. [Online]. Available: https://www.3ds.com/products-services/catia/products/dymola

[12] OpenModelica Project. (2021). *The OpenModelica Project Website*. Accessed: May 17, 2022. [Online]. Available: https://www.openmodelica.org/

[13] P. Fritzson, A. Pop, K. Abdelhak, A. Asghar, B. Bachmann, W. Braun, and D. Bouskela, "The OpenModelica integrated environment for modeling, simulation, and model-based development," *Model., Identificat. Control*, vol. 41, no. 4, pp. 241–295, 2020.

[14] A. Asghar, S. Tariq, M. Torabzadeh-Tari, P. Fritzson, A. Pop, M. Sjölund, P. Vasaiely, and W. Schamai, "An open source Modelica graphic editor integrated with electronic notebooks and interactive simulation," in *Proc. 8th Int. Modelica Conf.*, vol. 2011, pp. 739–747.

[15] K. Zakova, "Online use of OpenModelica via web service," in *Proc. 12th Int. Conf. Remote Eng. Virtual Instrum. (REV)*, Feb. 2015, pp. 152–156.

[16] K. Zakova and M. Cech, "Design of control education interactive examples via web service for OpenModelica," in *Proc. 13th APCA Int. Conf. Autom. Control Soft Comput. (CONTROLO)*, 2018, pp. 242–246.

[17] C. Martin-Villalba, A. Urquia, and S. Dormido, "Object-oriented modelling of virtual-labs for education in chemical process control," *Comput. Chem. Eng.*, vol. 32, no. 12, pp. 3176–3186, Dec. 2008.

[18] EJS. (2022). *Easy Java Simulations (EJS) Website*. Accessed: May 17, 2022. [Online]. Available: http://fem.um.es/Ejs/

[19] Sysquake. (2022). *Sysquake Website*. Accessed: May 17, 2022. [Online]. Available: https://calerga.com/

[20] (2022). *Functional Mock-up Interface*. Accessed: May 17, 2022. [Online]. Available: https://fmi-standard.org/

[21] X. Pang, R. Dye, T. S. Nouidui, M. Wetter, and J. J. Deringer, "Linking interactive Modelica simulations to HTML5 using the Functional Mockup Interface for the LearnHPB platform," in *Proc. 13th IBPSA Conf.*, 2013, pp. 2823–2829.

[22] V. Waurich and J. Weber, "Interactive FMU-based visualization for an early design experience," in *Proc. 12th Int. Modelica Conf.*, vol. 2017, pp. 879–885.

[23] V. Havard, B. Jeanne, M. Lacomblez, and D. Baudry, "Digital twin and virtual reality: A co-simulation environment for design and assessment of industrial workstations," *Prod. Manuf. Res.*, vol. 7, no. 1, pp. 472–489, 2019.

[24] B. Thiele, T. Beutlich, V. Waurich, M. Sjölund, and T. Bellmann, "Towards a standard-conform, platform-generic and feature-rich Modelica Device Drivers library," in *Proc. 12th Int. Modelica Conf.*, 2017, pp. 713–723.

[25] C. Höger, A. Mehlhase, C. Nytsch-Geusen, K. Isakovic, and R. Kubiak, "Modelica3D-platform independent simulation visualization," in *Proc. 9th Int. Modelica Conf.*, vol. 2012, pp. 485–494.

[26] DLR. (2022). *DLR Visualization Library*. [Online]. Available: https://www.systemcontrolinnovationlab.de/the-dlr-visualization-library/

[27] C. Martin-Villalba, A. Urquia, and S. Dormido, "Development of virtual labs for education in chemical process control using Modelica," *Comput. Chem. Eng.*, vol. 39, pp. 170–180, Apr. 2012.

[28] C. Martin-Villalba, A. Urquia, and S. Dormido, "An approach to virtual lab implementation using Modelica," *Math. Comput. Model. Dyn. Syst.*, vol. 14, no. 4, pp. 341–360, 2008.

[29] M. Association. (2017). *Modelica Specification, Version 3.4*. Accessed: May 6, 2022. [Online]. Available: https://modelica.org/documents/ModelicaSpec34.pdf

[30] G. Lazar and R. Penea, *Mastering Qt 5: Create Stunning Cross-Platform Applications Using C++ With Qt Widgets and QML With Qt Quick*. Birmingham, U.K.: Packt, 2018.

[31] QWT. (2022). *The QWT User's guide*. Accessed: May 17, 2022. [Online]. Available: http://qwt.sourceforge.net

[32] W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit an Object-Oriented Approach To 3D Graphics Edition 4.1*. Kitware, Clifton Park, NY, USA, 2018.

[33] *The Visualization Toolkit User's Guide*. Kitware, Clifton Park, NY, USA, 2010.

[34] J. J. Downs and E. F. Vogel, "A plant-wide industrial process control problem," *Comput. Chem. Eng.*, vol. 17, no. 3, pp. 245–255, Mar. 1993.

[35] K. J. Åström and T. Hagglund, *PID Controllers: Theory, Design and Tuning*. Research Triangle, NC, USA: ISA Press, 1995.

CARLA MARTIN-VILLALBA received the degree in electronic engineering from the Universidad Complutense de Madrid, Spain, in 2001, and the Ph.D. degree in computer science from the Universidad Nacional de Educación a Distancia (UNED), Madrid, Spain, in 2007. She is currently an Associate Professor with the Departamento de Informática y Automática, UNED. Her research interests include modeling and simulation, automatic control, and distance learning.

ALFONSO URQUIA received the M.S. degree in physics from the Universidad Complutense de Madrid, in 1992, and the Ph.D. degree in physics from the Universidad Nacional de Educación a Distancia (UNED), Madrid, Spain, in 2000. Since 2002, he has been working as an Associate Professor with the Departamento de Informática y Automática, UNED. His research interests include mathematical modeling and computer simulation.