Computers and Chemical Engineering 32 (2008) 3176–3186

Contents lists available at ScienceDirect

Computers and Chemical Engineering

journal homepage: www.elsevier.com/locate/compchemeng

Object-oriented modelling of virtual-labs for education in chemical process control

Carla Martin-Villalba*, Alfonso Urquia, Sebastian Dormido

Dept. Informática y Automática, UNED, Juan del Rosal 16, 28040 Madrid, Spain

ARTICLE INFO

Article history:
Received 27 January 2006
Received in revised form 9 September 2007
Accepted 19 May 2008
Available online 7 July 2008

Keywords:
Control education
Virtual laboratory
Interactive simulation
Chemical process control
Object-oriented modelling
Modelica

ABSTRACT

Easy Java Simulations (Ejs) and Sysquake are two software tools specifically intended for implementation of virtual-labs. They allow easy definition of the virtual-lab *view* (i.e., the model-to-user interface). However, the *model* definition capabilities and the numerical solvers provided by these tools are not the state-of-the-art.

On the other hand, the use of the object-oriented modelling language Modelica reduces considerably the modelling effort and permits better reuse of the models. Modelica is supported by the state-of-the-art simulation environment Dymola. Nevertheless, Modelica does not provide the interactive capabilities required for virtual-lab implementation.

The approach proposed in this manuscript is to combine the best features of each tool. Ejs and Sysquake capability for building interactive user-interfaces composed of graphical elements, whose properties are linked to the model variables. Modelica capability for physical modelling and Dymola capability for simulating DAE-hybrid models. This novel approach has been successfully applied to set up virtual-labs for control education.

1. Introduction

Virtual-labs are effective educational tools for training of process engineers and plant operators. They provide a flexible and user-friendly method to define the experiments to be performed on the mathematical model (Jimoyiannis & Komis, 2001). Virtual-labs allow users to design and perform their own simulation experiments. As a result, users become active players in their own learning process, which motivate them to learn.

Virtual-labs are composed of: (1) the simulation of the *mathematical* model describing the relevant properties of the system; (2) the interactive user-to-model interface, referred to as virtual-lab *view*; (3) a narrative that provides information about the system and the use of the virtual-lab.

The virtual-lab *view* is intended to provide a visual representation of the model dynamic behaviour and to facilitate the user's interactive actions on the model. The graphical properties of the view elements are linked to the model variables, producing a bidirectional 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 model behaviour can be represented in different ways. For instance, plotting the model variables against each other and by means of animated schematic diagrams of the system. In addition, linear systems can be described using pole-zero diagrams and frequency response diagrams (i.e., Bode and Nyquist diagrams). User's actions on the model can be performed by manipulating different elements of the view, such as buttons, sliders, check-boxes and certain graphic elements of the model schematic diagram.

1.1. Types of interactivity

In some cases, the virtual-lab is intended to emulate the real-time response of the plant, allowing the user to make real-time decisions. In these kind of applications, the user is allowed to perform interactive actions on the model at any time during the simulation run. The user can change the value of the model inputs, parameters and state variables, 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. This type of interactivity is called *runtime interactivity*.

In other cases, the goal is to obtain the simulated response of the model for a specific time period, and use it to automatically compute linear approximations to the system and their frequency-domain characteristics, to perform automatic synthesis of controllers, etc. In this other kind of applications, the user's action triggers the start of the simulation, which is run to completion.

* Corresponding author. Tel.: +34 913988253.
E-mail address: carla@dia.uned.es (C. Martin-Villalba).

0098-1354/$ – see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compchemeng.2008.05.011

During the simulation run, the user is not allowed to interact with the model. Once the simulation run is finished, the results are displayed and a new user's action on the model is allowed. This type of interactivity is called *batch interactivity*. The implementation of virtual-lab with runtime and with batch interactivity is discussed in this manuscript.

1.2. Virtual-lab implementation software

Several virtual-lab packages, conceived to illustrate some selected topics in automatic control, have been implemented. ICTools and CCSDEMO (Johansson, Gäfvert, & Aström, 1998; Wittenmark, Häglund, & Johansson, 1998) are two packages developed at the Department of Automatic Control, Lund Institute of Technology. Packages especially designed for chemical process control education have been developed by (Cooper & Dougherty, 2000; Cooper, Dougherty, & Rice, 2003; Doyle, Gatzke, & Parker, 1998; Young, Mahoney, & Svrcek, 2001). Matlab/Simulink is widely used for the development of virtual-labs in the context of control education. This modelling and simulation environment supports the *graphical block diagram modelling*. Other virtual-lab implementations use different software tools, such as the process modelling tool Hysys.

On the other hand, there are software tools specifically intended for implementation of virtual-labs. These tools: (1) provide their own procedures to define the narrative, the model and the view of the virtual-lab; (2) guide the virtual-lab programmer in these tasks; (3) automatically generate the virtual-lab executable code. Easy Java Simulations and Sysquake are two of these tools.

Sysquake (Calerga, 2004; Piguet, Holmberg, & Longchamp, 1999) is a Matlab-like environment for developing virtual-labs with batch interactivity. Typically, a Sysquake application contains several interactive graphics, which are displayed simultaneously. These graphics contain elements that can be manipulated using the mouse. While one of these elements is being manipulated, the other graphics are automatically updated to reflect this change. The content represented by each graphic, and its dependence with respect to the content of the other graphics, is programmed using LME (an interpreter for numerical computation which is mostly compatible with Matlab). Sysquake can be extended by plug-ins and libraries of functions written in LME.

Easy Java Simulations (Esquembre, 2004) is an open source, Java-based software tool intended to implement virtual-labs with runtime interactivity. It can be freely downloaded from http://fem.um.es/Ejs/. Ejs guides the user in the process of creating the narrative, the model and the view of the virtual-lab. It generates automatically: (1) the interactive simulation as a Java application; (2) HTML pages containing the narrative and the interactive simulation as a Java applet. Then, the user can run the virtual-lab and/or publish it on the Internet. Ejs allows the user to include new Java classes.

A strong point of these tools is that they allow easy definition of the virtual-lab view. Easy Java Simulations (hereafter cited as Ejs) provides a complete set of interactive graphic elements which are ready to be used, in a simple drag-and-drop way, to compose the view. Additionally, it facilitates the integration of multimedia elements such as video and sound. Sysquake supports built-in functions to include in the view different types of interactive plots and interactive graphic elements (i.e., radio-buttons, sliders, dialog boxes, etc.).

However, the model definition capabilities and the numerical solvers provided by these tools are not the state-of-the-art. They support the *block diagram modelling*. This modelling paradigm requires of explicit state models (i.e., ordinary differential equations) and the computational causality of the model must be explicitly set (i.e., the blocks have a unidirectional data flow from inputs to outputs). These restrictions do not facilitate the model reuse and they strongly condition the modelling task, which requires a considerable effort. For instance, dummy dynamics need to be introduced in the model to avoid the establishment of systems of simultaneous equations. The model programmer has to manipulate the model to transform its equations to the form of ordinary differential equations (ODE). As a consequence, the modelling and simulation capabilities supported by these tools are not the best possible ones for describing the large models used in the physical–chemical field.

The *physical modelling paradigm*, supported by the object-oriented modelling languages, is an attractive alternative to the *block diagram modelling* (Aström, Elmqvist, & Mattsson, 1998). Object-oriented modelling languages support a declarative description of the model, based on equations instead of assignment statements. The computational causality is not included in the model. Thus, a model can adapt to more than one data flow context. The modelling knowledge is represented as differential, algebraic and discrete equations that may change by being triggered by events (i.e., hybrid-DAE models).

Some object-oriented modelling languages are EcosimPro Language, gPROMS and Modelica. The two first are commercial languages, whereas Modelica is a free language. The Modelica language has been designed by the developers of the object-oriented modelling languages Allan, Dymola, NMF, ObjectMath, Omola, SIDOPS+, Smile and a number of modelling practitioners in different fields. It is intended to serve as a standard format for the external model representation, so that models arising in different engineering fields can be exchanged between tools and users.

Currently, a number of free and commercial component libraries in different domains are available (http://www.modelica.org/), including electrical, mechanical, thermo-fluid and physical–chemical. Modelica is well suited for describing the type of multi-domain models used in automatic control. Modelica language is supported by several simulation environments, including Dymola (Dynasim, 2002).

The use of the Modelica language reduces considerably the modelling effort and permits better reuse of the models. Dymola is the state-of-the-art simulation environment supporting Modelica. During the model translation, Dymola carries out the manipulations intended to transform the model into an efficiently solvable form. As a result, it generates highly efficient C code. Previous efforts on virtual-lab implementation using Modelica language include (Engelson, 2000; Martin, Urquia, & Dormido, 2007).

1.3. Contributions of this paper

The approach proposed in this manuscript is to combine the best features of each tool. Ejs and Sysquake capability for building interactive user-interfaces composed of graphical elements, whose properties are linked to the model variables. Matlab/Simulink capability for modelling of automatic control systems and for model analysis. Modelica capability for physical modelling, and finally Dymola capability for simulating hybrid-DAE models. This software combination approach is discussed for implementation of virtual-labs with *runtime interactivity* (Section 2) and with *batch interactivity* (Section 3).

This novel approach has been applied to the implementation of a set of virtual-labs that are being used for process control education at the National University of Distance Education of Spain (UNED). These virtual-labs are concerned with some key aspects in process control education (Perkins, 2002), illustrating a range of topics that should be mastered by process system engineers (Johansson