# INTERACTIVE SIMULATION OF OBJECT-ORIENTED HYBRID MODELS, BY COMBINED USE OF EJS, MATLAB/SIMULINK AND MODELICA/DYMOLA

Carla Martin
Alfonso Urquia
Jose Sanchez
Sebastian Dormido
Dept. Informática y Automática
E.T.S. Ingeniería Informática, UNED
Juan del Rosal 16, 28040 Madrid, Spain
{carla,aurquia,jsanchez,sdormido}@dia.uned.es

Francisco Esquembre
Dept. Matemáticas
Universidad de Murcia
Campus de Espinardo,
30071 Murcia, Spain
fem@um.es

Jose Luis Guzman
Manuel Berenguel
Dept. Lenguajes y Computación
Universidad de Almería
Ctra. Sacramento s/n,
04120 Almería, Spain
{joguzman,beren}@ual.es

**KEYWORDS**

Interactive simulation, web-based simulation, object-oriented modeling, hybrid models, quadruple-tank process, Easy Java Simulations.

## ABSTRACT

Easy Java Simulations (Ejs) is a freeware, open source, Java-based tool intended to create interactive dynamic simulations. The use of Ejs, together with Matlab/Simulink and Modelica/Dymola allow us to combine the best features of each tool. Ejs capability for building interactive user-interfaces composed of graphical elements, whose properties are linked to the model variables. Matlab/Simulink capability for modeling of automatic control systems and for model analysis. Modelica capability for physical modeling, and finally Dymola capability for simulating hybrid-DAE models. The combined application of these tools to the implementation of interactive simulations is discussed in this manuscript, and a novel modeling methodology adequate for interactive simulation is proposed. It takes advantage of the modeling and simulation capabilities of Modelica and Dymola. The proposed methodology is successfully applied to a case study: the interactive simulation of the quadruple-tank process.

## INTRODUCTION

Interactive simulation provides a flexible and user-friendly method to define the experiments performed on the model. During the interactive 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. As a consequence, interactive simulation facilitates developing and enhancing the understanding of the model behavior. This capability is especially useful when the model is being used for educational purposes.

Easy Java Simulations (Ejs) is a freeware, open source, Java-based tool intended to create interactive dynamic simulations (Esquembre, 2004; http://fem.um.es/Ejs/). Ejs was originally designed to be used by students for interactive learning, under the supervision of educators with a low programming level. As a consequence, simplicity was a requirement. Ejs guides the user in the process of creating interactive simulations. This process includes the definition of the *model* and the *view*.

Ejs implements its own procedure to define the *model*: a simple structure that the user needs to complete in order to specify the model variables and equations. In addition, Ejs version 3.2 supports the option of describing and simulating the model using Matlab/Simulink: (1) Matlab code and calls to any Matlab function (either built-in or defined in an M-file) can be used at any point in the Ejs model; and (2) the Ejs model can be partially or completely developed using Simulink block diagrams.

The *view* is the user-to-model interface of Ejs interactive simulations. It is intended to: (1) provide a visual representation of the model relevant properties and dynamic behavior; and (2) facilitate the user's interactive actions on the model. Ejs includes a set of ready-to-use visual elements, that the modeler can use to compose a sophisticated view in a simple, drag-and-drop way. The properties of the view elements can be 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.

Once the modeler has defined the model and the view of the interactive simulation, Ejs generates the Java source code of the simulation program, compiles the program, packs the resulting object files into a compressed file, and generates HTML pages containing the narrative and the simulation as an applet. The user can readily run the simulation and/or publish it on the Internet.

Working together with Matlab/Simulink significantly improves the Ejs capabilities for model description and numerical solution. However, Simulink modeling paradigm (i.e., graphical block-diagram modeling) exhibits some limitations (Astrom et al. 1998). It requires explicit state models (ODE) and that the blocks have a unidirectional data flow from inputs to outputs. These restrictions strongly condition the modeling task, which requires a considerable effort from the modeller.

Object-oriented modeling languages facilitate the physical modeling paradigm (Astrom et al. 1998), an attractive alternative to the block diagram modeling. They support a declarative (non-causal) description of the model, which permits better reuse of the models. The modeling knowledge is represented in terms of hybrid-DAE models: differential, algebraic and discrete equations that may change by being triggered by events. As a consequence, the use of object-oriented modeling languages reduces considerably the modeling effort. In order to achieve this goal, the use of Modelica/Dymola together with Ejs and Matlab/Simulink is proposed in this manuscript, in addition to a novel modeling methodology intended for interactive simulation. This methodology is successfully applied to a case study: the interactive simulation of the quadruple-tank process.

## COMBINED USE OF EJS, MATLAB/SIMULINK AND MODELICA/DYMOLA

Dymola 5.0 interface to Matlab 5.3 / Simulink 3.0 can be found in Simulink's library browser: DymolaBlock block (Dynasim 2002). This block is an interface to the C-code generated by Dymola for the Modelica model. This C-code contains the numerical algorithms required to simulate the Modelica model, which is formulated in terms of differential-algebraic equations (DAE) and discrete-time equations. DymolaBlock block solution is synchronized with the solution of the other blocks: Simulink integrates DymolaBlock blocks together with the other blocks.

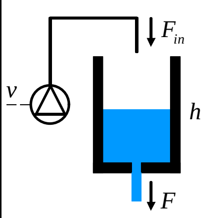
Double-clicking on the DymolaBlock block opens a form where the name and file of the Modelica model name are specified. In order to embed the Modelica model within a Simulink block, the computational causality of the Modelica model interface needs to be explicitly set. The input variables are supposed to be calculated from other blocks, while the output variables are calculated from the Modelica model. DymolaBlock block can be connected to other Simulink blocks and also to other DymolaBlock blocks.

Ejs 3.2 supports the option of describing and simulating some parts of the model (or the complete model) using Matlab/Simulink. Ejs synchronizes the numerical solution of the model part described using Ejs procedure and the part described in terms of Simulink blocks. In order to simulate the Ejs part of the model, Ejs implements a set of built-in ODE solvers, and it allows the modeler to program and use his own numerical algorithms. Matlab code is calculated in the Matlab workspace. Simulink part of the model is simulated by Matlab/Simulink, using Simulink's numerical algorithms.

## MODELING FOR INTERACTIVE SIMULATION USING MODELICA LANGUAGE

A novel modeling methodology intended for interactive simulation is proposed. It takes advantage of the modeling and simulation capabilities of Modelica and Dymola. The proposed methodology takes into account the common characteristics of the models intended for interactive simulation. As a result of the user interaction, these models need to support instantaneous (discontinuous) changes in the state, and in the value of the input variables and parameters. In addition, several choices of the state variables need to be simultaneously supported, in order to provide alternative ways of describing the state changes.

The model shown in Fig. 1 will be used to illustrate the discussion. The voltage applied to the pump (*v*) is an input variable. The cross-sections of the tank (*A*) and the outlet hole (*a*), the pump parameter (*k*) and the gravitational acceleration (*g*) are time-independent properties of the physical system.



$$\frac{dV}{dt} = F_{in} - F$$
$$F = a\sqrt{2gh}$$
$$V = Ah$$
$$F_{in} = kv$$

*V*: liquid volume
*h*: liquid level
*a*: hole cross-section
*A*: tank cross-section
*F*, *F<sub>in</sub>*: liquid flow
*g*: grav. acceleration
*v*: pump input voltage
*k*: pump parameter

Figure 1: Model of a Process

### Interactive Changes on the Model State

Two input variables to the DymolaBlock block are used to model the interactive changes in the state: *Istate*[:] and *CKstate* (see Fig. 2). The array *Istate* contains the values used to reinitialize the model state. The signal *CKstate* is used to trigger the state re-initialization event. The state re-initialization is performed using a built-in Modelica operator: *reinit*(*x*,*expr*). It re-initializes an state variable (*x*) with the value obtained of evaluating an expression (*expr*), at an event instant. The output-variable array of the DymolaBlock block (*O*[:]) contains the variables representing the actual value of the state, in addition to the other variables linked to the view elements. Ejs uses the value of this output array (*O*[:]) to refresh the simulation view. When the user changes the value of one or several state variables, Ejs writes the new state in *Istate* array and changes the value of *CKstate* (from one to zero or from zero to one) in order to trigger the re-initialization event.

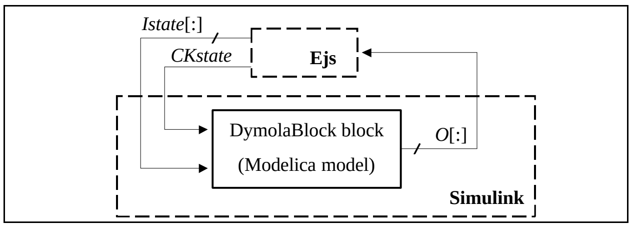


Figure 2: Interactive Change of the State

Different choices of the model state-variables are possible. For instance, possible choices in the model of Fig. 1 include: $e_1 = \{h\}$, $e_2 = \{V\}$ and $e_3 = \{F\}$; where $e_i$ represents one particular choice of the state variables. 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 *V*, then the right choice is $e_2$; and if he wants to change *F*, then $e_3$. The model should support the following feature: every time the user needs to change the state value, he decides to represent it in terms of a change in either the volume or the height or the flow. Different choices are possible during a given interactive simulation run.

In general, an interactive model is required to support changes in its state value that correspond with different choices of the state variables. An approach to fulfill this requirement is the following. Modeling 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 allows the implementation of this approach conveniently. In addition, the interactive model needs to describe the information flow among these different instantiations of the physical model. These topics will be discussed later.

## Changes on the Interactive Parameters

Time-independent properties of the system are usually represented by model parameters. However, sometimes one of the interactive-simulation goals is studying the dependence between the model dynamic behavior and the value of these properties. In this case, the property value can be instantaneously changed by the user's action, remaining constant between consecutive interactive changes. These variables of the interactive model are called *interactive parameters*. DymolaBlock block inputs should include the variables required to describe the changes in the interactive-parameter values.

Changes in the value of interactive parameters can have different effects depending on the state variable choice. Consider an instantaneous change in the cross-section (*A*) of the model in Fig. 1 (for instance, the user interactively changes *A* from $1\,\mathrm{cm}^2$ to $3\,\mathrm{cm}^2$). If the state variable is the volume (*V*), then the change in $A$ produces an instantaneous change in the value of the liquid height (*h*) and flow (*F*), while the liquid volume remains constant. On the contrary, if the state variable is the height or the flow, these magnitude values do not change as the result of an instantaneous change in $A$. In this case, the volume does change.

In general, interactive models need to support instantaneous changes in the value of the interactive parameters, for different choices of the state variables. This requirement can be accomplished by using the adequate physical-model instantiation (that with the required state selection) for executing the instantaneous change in the parameter value 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.

## State Selection Control

Modelica 2.0 supports the user's control on the state variables selection, via the *stateSelect* attribute of Real variables (Otter and Olsson 2002). This attribute values include "never" (the variable will never be selected as state variable) and "always" (the variable will always be used as a state). This Modelica feature allows controlling the model state selection by means of a Boolean array. State selection of the model in Fig.1 can be accomplished as shown below, by means of the Boolean vector *isState*. For instance, if *isState* is set to the value {false,true,false} when instantiating the physical-model, then the volume is selected as a state variable.

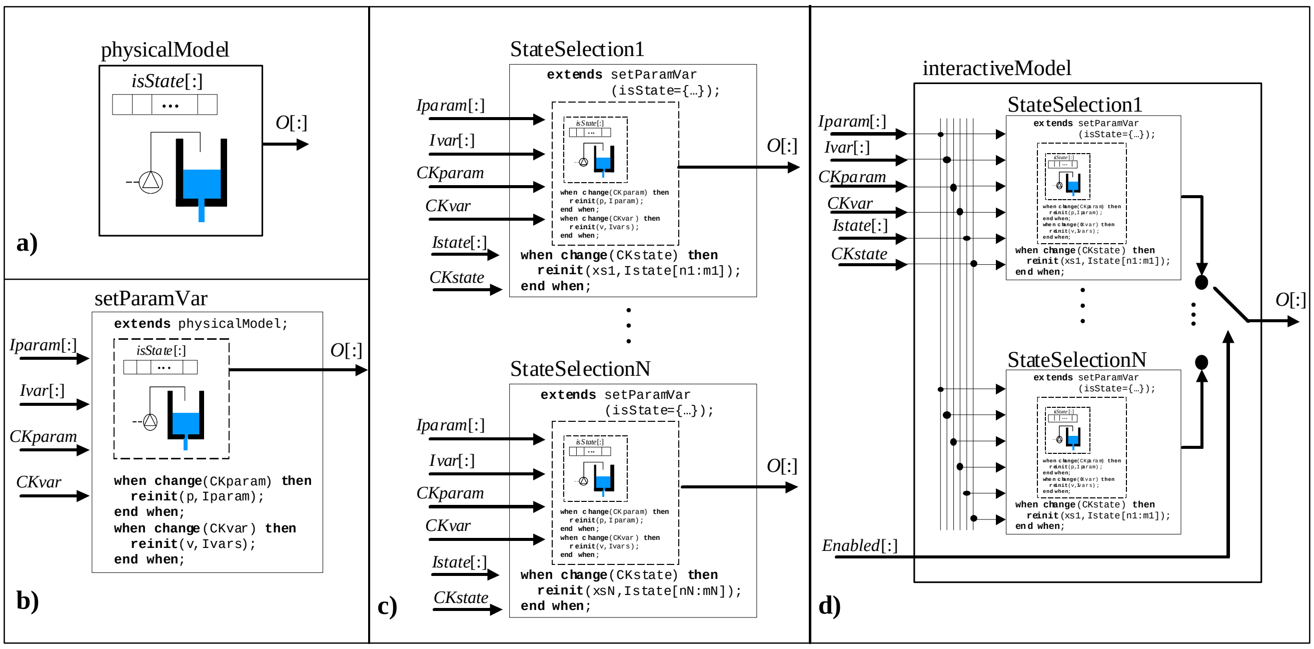
```
model tank
  Real h (stateSelect = if hIsState
     then StateSelect.always else StateSelect.never);
  Real volume (stateSelect = if volumeIsState
     then StateSelect.always else StateSelect.never);
  ...
end tank;

model pipe
  Real F (stateSelect = if FIsState
     then StateSelect.always else StateSelect.never);
...
end pipe;

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

## Conceptual Foundations of State Selection Control

Dymola 5.1 (Mattsson et al. 2000; Dynasim 2002) supports this Modelica capability: the user's control on the state-variable selection. The continuous part of a Modelica model is mapped to a DAE of the form $\mathbf{f}(\dot{\mathbf{x}}_1, \mathbf{x}_1, \mathbf{x}_2, t) = \mathbf{0}$, where $\mathbf{x}_1(t)$ are variables appearing differentiated, and $\mathbf{x}_2(t)$ are pure algebraic variables. Conceptually, once the state-variables (i.e., $\mathbf{x}_S(t)$) have been selected, Dymola transforms automatically this DAE into the state space form:

$$\begin{aligned} \dot{\mathbf{x}}_S &= \mathbf{f}_1(\mathbf{x}_S, t) \\ \mathbf{x}_n &= \mathbf{f}_2(\mathbf{x}_S, t) \end{aligned} \qquad (1)$$

The tank model shown in Fig. 1 will be used to illustrate the conceptual foundations of this transformation. According to the model description in Fig. 1: $\mathbf{x}_1 = \{V\}$.



```
extends physicalModel;

when change(CKparam) then
  reinit(p, Iparam);
end when;
when change(CKvar) then
  reinit(v, Ivars);
end when;
```

```
extends setParamVar
        (isState={…});

when change(CKstate) then
  reinit(xs1, Istate[n1:m1]);
end when;
```

```
extends setParamVar
        (isState={…});

when change(CKstate) then
  reinit(xsN, Istate[nN:mN]);
end when;
```

Figure 3: Schematic Description of the Proposed Modeling Methodology for Interactive Simulation

In order to reformulate the model as shown in Equation (1), with $\mathbf{x}_S = \{h\}$, a dummy variable $a$ and an equation ($\dot{h} = a$) are included in the model (H. Elmqvist: personal communication, 1995). These changes do not modify the model: the equation $\dot{h} = a$ is used to calculate $a$. The index of the extended model is greater than one (Brenan et al. 1996). The model index can be reduced to one or zero by applying the Pantelides algorithm (Pantelides 1998) and by including the differentiated equations in the model (Mattsson and Soderlind 1992). Then, the dummy derivatives are defined so that the user's state selection is fulfilled. The result of these manipulations is shown below, where the variable to evaluate from each equation is written within square brackets.

$$\begin{aligned}\dot{V} &= F_{in} - F\\ F &= a\sqrt{2gh}\\ V &= Ah\\ F_{in} &= kv\\ \dot{h} &= a\end{aligned} \quad\longrightarrow\quad \begin{aligned}[F] &= a\sqrt{2gh}\\ [F_{in}] &= kv\\ [derV] &= F_{in} - F\\ [V] &= Ah\\ derV &= \dot{A}h + A\left[\dot{h}\right]\\ \dot{h} &= [a]\end{aligned}$$

## Interactive Parameters and Input Variables

This method to formulate the model according to a given state selection can require differentiating an interactive parameter or an input variable. In the previous example, the time-derivative of the tank cross-section needs to be calculated. If the interactive parameter $A$ is defined as an input variable, then an error is produced: Dymola cannot differentiate an input variable.

A valid approach consists in defining the interactive parameters as constant state-variables (i.e., with zero time-derivative). The interactive changes in the value of these parameters are implemented by re-initializing their values. An analogous reasoning is applicable to the input variables of the physical model, and an analogous solution is implemented. Four input variables to the DymolaBlock block are used to model the changes in the value of the interactive parameters and input variables: two signal arrays (*Iparam*[:], *Ivar*[:]) containing the new values, and two signals (*CKparam* and *CKvar*) for triggering the re-initialization events.

## Modeling Methodology for Interactive Simulation

Summarizing the previous discussion, the proposed methodology consists in the following steps:

1. Object-oriented modeling of the physical system: model *physicalModel*. The Boolean vector *isState* controls the state selection (see Fig. 3a).
2. Define the model *setParamVar* (see Fig. 3b). It inherits *physicalModel* and contains the when-clauses to change the value of the interactive parameters (*p*[:]) and input variables (*v*[:]).
3. Define as many models (*stateSelection1*, *stateSelection2*, ...) as different state-variable choices needed ($e_1$, $e_2$, ...). Each of these models inherits *setParamVar* (*isState* array is set to the value adequate in each case) and contains a when-clause to change the value of the corresponding state-variable array (see Fig. 3c).
4. Define the model *interactiveModel*, composed of all the models defined in the previous step (*stateSelection1*, ..., *stateSelectionN*). The value of the input array *Enabled*[1:*N*] is set by Ejs, and it selects which output is connected to the output signal (*O*[:]).

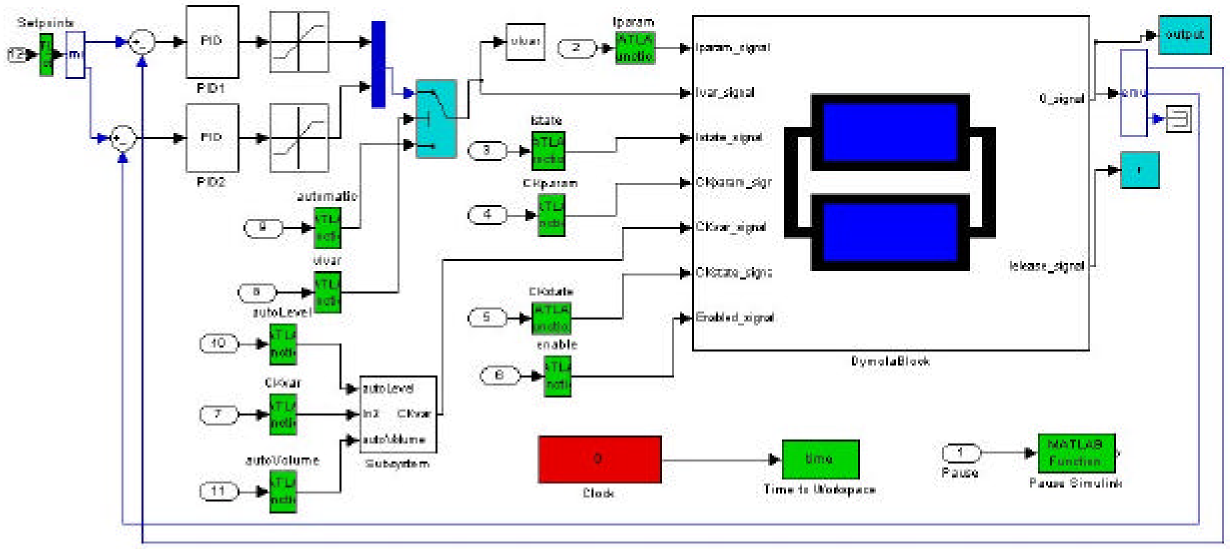


Figure 4: Simulink Model of the Quadruple-Tank Process and its Control System

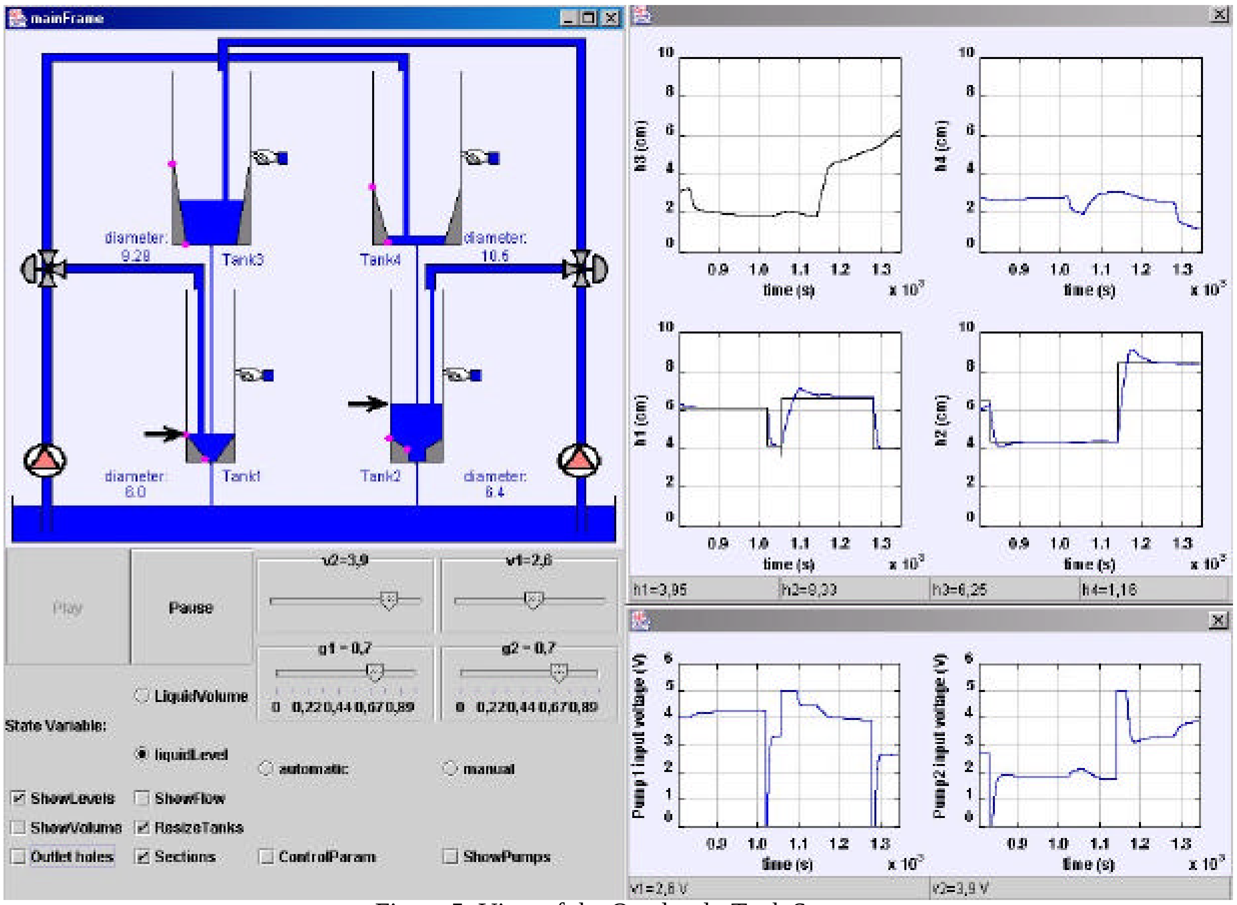


Figure 5: View of the Quadruple-Tank System

The obtained Modelica model is schematically represented in Fig. 3d. It is suitable to be embedded in a DymolaBlock block and connected to Ejs for interactive simulation.

## CASE STUDY: INTERACTIVE SIMULATION OF THE QUADRUPLE-TANK PROCESS

The interactive simulation of the quadruple-tank process has been implemented (see Fig. 5), by the combined use of Ejs, Matlab/Simulink and Modelica/Dymola. This process has been used to illustrate different aspects of the multi-variable control theory (Johansson 2000; Dormido and Esquembre 2003): the goal is to control the level of the lower two tanks with two pumps. Two different control strategies have been implemented in this case study: manual control and decentralized PID. Switching between these control strategies can take place during the simulation run. The parameters of the PID controllers can be changed interactively.

The simulation supports interactive changes in the tank physical parameters: cross-section, shape and cross-section of the outlet hole. Tank cross-section and shape changes can take place under one of two alternative conditions: the liquid volume inside the tank is kept constant or the liquid height is kept constant. In addition, the simulation supports interactive changes in the amount of liquid stored inside the tanks. These changes can be defined in terms of the liquid height or the liquid volume.

The physical model of the quadruple-tank process has been implemented in Modelica language. It is analogous to the model shown in Fig. 1. Mass balance and Bernoulli's law are applied to model the tanks and the flows. The process model (DymolaBlock block) and the control system are shown in Fig. 4. The automatic control system consists in two PID blocks followed by saturators. Depending on a variable that control a switch, the pump voltage values sent to Dymolablock are the ones obtained by PID controllers or the ones introduced by the user.

The simulation view is shown in Fig. 5. The main window (on the left side of Fig. 5) contains the schematic diagram of the process (above) and the control buttons (below). Both of them allow the user to experiment with the model. The liquid levels, the tank cross-sections and the level setpoints represented in the schematic diagram are linked to the respective model variables: their values can be interactively changed by dragging with the mouse. The sliders placed under the schematic diagram allow changing interactively the PID parameters and the cross-sections of the outlet holes. The control buttons allow choosing the state variables (liquid volumes or heights). They also open and close graphic plots of the liquid levels, volumes and flows, and plots of the voltages applied to the pumps. Some of these plots are displayed on the right side of Fig. 5.

## CONCLUSIONS

The feasibility of combining Ejs, Matlab/Simulink and Modelica/Dymola for implementing interactive simulations has been discussed. This approach takes advantage of Ejs capability for building interactive user-interfaces, Matlab/Simulink capability for modeling of automatic control systems, Modelica capability for physical modeling, and Dymola capability for simulating hybrid-DAE models. In addition, a novel modeling methodology adequate for interactive simulation has been proposed, and it has been successfully applied to a case study: the interactive simulation of the quadruple-tank process.

## REFERENCES

Astrom, K.J.; H. Elmqvist; and S.E. Mattsson. 1998. "Evolution of Continuous-Time Modeling and Simulation". In *Proceedings of the 12th European Simulation Multiconference* (Manchester, UK).

Dormido, S. and F. Esquembre. 2003. "The Quadruple-Tank Process: An Interactive Tool for Control Education", In *Proceedings of the 2003 European Control Conference*, (Cambridge, UK).

Dynasim. 2002. "Dymola. User's Manual. Version 5.0a". Dynasim AB. Lund, Sweden.

Esquembre, F. 2004. "Easy Java Simulations: a software tool to create scientific simulations in Java" *Computer Physics Communications*, 156, 199-204.

Johansson, K.H. 2000. "The Quadruple-Tank Process: A Multivariable Laboratory Process with an Adjustable Zero". *IEEE Transactions on Control Systems Technology*, Vol. 8, No. 3 (May), 456-465.

Mattsson, S.E.; H. Olsson; and H. Elmqvist. 2000. In *Proceedings of the Modelica Workshop 2000*, (Lund, Sweden), pp. 61 - 67.

Otter, M. and H. Olsson. 2002. "New features in Modelica 2.0". In *Proceedings of the 2nd International Modelica Conference*, (Oberpfaffenhofen, Germany), 7.1 - 7.12.

## AUTHOR BIOGRAPHIES

**CARLA MARTIN** received her M.S. degree in Electrical Engineering in 2001 from University Complutense of Madrid (UCM), Spain. She is currently completing her doctorate studies at UNED, Spain. (carla@dia.uned.es)

**ALFONSO URQUIA** was born in Madrid (1969). He received his M.S. degree in Physics in 1992 from UCM and his Ph.D. in 2000 from UNED. His work experience includes six years as an R&D engineer at AT&T and Lucent Technologies - Bell Labs. Currently he is working as an associate professor at UNED. (aurquia@dia.uned.es)

**JOSE SANCHEZ** received his Computer Sciences degree in 1994, from Madrid Polytechnic University and his Ph.D. in Sciences from UNED in 2001. Since 1993, he has been working at UNED Department of Computer Sciences and Automatic Control as an Assistant Professor. (jsanchez@dia.uned.es)

**SEBASTIAN DORMIDO** received his Physics degree from UCM (1968) and his Ph.D. from Country Vasc University (1971). In 1981, he was appointed Full Professor of Control Engineering at UNED Faculty of Sciences. Since 2002, he is President of CEA-IFAC. (sdormido@dia.uned.es)

**FRANCISCO ESQUEMBRE** was born in Alicante (1963) and received the Ph.D. degree in Mathematics in June 1991, from the University of Murcia, Spain, where he works since 1986, holding a permanent job as Assistant Professor since 1994. (fem@um.es)

**JOSE L.GUZMAN** was born in Almería, Spain. He received his Computer Science degree in 2002 from the University of Almería. Currently, he is working in his Ph.D at the University of Almeria. (joguzman@ual.es)

**MANUEL BERENGUEL** is associate professor in the University of Almería, Spain. (beren@ual.es)