Modeling of hydraulic systems for hardware-in-the-loop simulation: a methodology proposal

Jorge A. Ferreira
Department of Mechanical Engineering
University of Aveiro
3810 Aveiro, Portugal

João E. de Oliveira
Department of Electronic Engineering
University of Aveiro
3810 Aveiro, Portugal

Vítor A. Costa
Department of Mechanical Engineering
University of Aveiro
3810 Aveiro, Portugal

Abstract

The present paper proposes a methodology to organize model libraries of electro-hydraulic components. This methodology holds in the association of an object oriented modeling, equation based, language for model structure description, with a graphical formalism suitable for dynamic behavior description of reactive hybrid systems. A recent general purpose language for physical modeling, called Modelica [Mattsson and Elmqvist, 1997], is used to develop object-oriented libraries of models for different physical domains, along with the formalism of Statecharts [Harel, 1987] for hierarchical description of the model’s dynamic behavior. Complex models could then be built by model interconnection, being also possible to organize models with different complexities, appropriated for different simulations (real time or off-line simulations), refining the behavior using the Statecharts graphical formalism.

The compilation of the hydraulic model generates code in C language that can be executed, afterwards, on a digital signal processing card (DSP). The hardware-in-the-loop simulation will be implemented with the “virtual” hydraulic application being controlled by real controllers.

Keywords: modeling and simulation, electro-hydraulic systems, modeling language, graphical formalism, hardware-in-the-loop simulation.

1. Introduction

The hydraulic systems have been, for a long time, often used in industrial manufacturing and in heavy machinery. The hydraulic hardware suffered a great evolution during last years, from hydro-mechanical devices to sophisticated electro-hydraulic systems controlled by microprocessors. The use of electronics and microprocessors contributes to improve the dynamic performance and to increase the traditional systems with new features, as well as with new control possibilities. Due to the complex dynamics and non-linearity of these type of systems, the control algorithms usually applied in linear systems, such as the traditional PID algorithms, can have a poor performance for sophisticated hydraulic applications. Nowadays, there is an increased interest on strategies for using electro-hydraulics in advanced manufacturing systems, where dynamic performance and precision are very important parameters as, for example, in high-speed machining, injection molding systems or high-speed assembly hydraulic robots. It is important to investigate how advanced control schemes can improve the hydraulic actuation of this type of machinery.

Accordingly to [Edge, 1997], it is important that, for a given application, the relative merits of different control schemes can improve the hydraulic actuation of this type of machinery. Accordingly to [Edge, 1997], it is important that, for a given application, the relative merits of different control schemes can be evaluated, being the computer simulation one of the best evaluation tool. Supporting this idea is [Ellman et al., 95], referring that a simulated environment is the cheapest and fastest way to test control algorithms. Modeling and real time simulation of complex systems still is an area to explore [Burrows, 1998] and, with the growing of computation power, more
and more complex systems can be simulated in real time with decreasing costs [Lennev et al., 1995].

Application of new control schemes on real hydraulic systems is, a difficult task due to the cost and/or size of hardware and its working conditions. In many applications, it is impossible to reproduce, in the laboratory, the hydraulic systems and their operating conditions. Some specific studies have been performed, where the operating environment and the hydraulic machinery hardware, were reproduced by simulation and 3D visualization of the simulated model’s dynamic behavior. Examples of such studies can be found in [Gonthier and Papadopoulos, 1998], [DiMaio at al., 1998] or [Schothorst, 1997]. However, these works uses private modeling methodologies, thus precluding the interchange and the refining of models.

2. Why a new methodology proposal?

Although the strong recent evolution in electro-hydraulic hardware, the project of hydraulic systems still is essentially based on tradition and experience. This fact, added to the growing complexity of modern hydraulic systems, can lead to unexpected behavior and errors. One way to predict these situations is the use of computer simulation, in many cases with ad-hoc simulation programs, usually written in FORTRAN, or with physical prototypes [Ellman et al., 1995].

The various physical domains involved (hydraulics, mechanics, electrics, etc.) are an additional complexity. Another problem is the dynamic behavior of involved components which can have a continuous and/or discrete dynamic behavior (hybrid systems), thus requiring that the selected modeling language has to deal with various physical domains and hybrid systems’ modeling.

The representation of models in a systematic and flexible way has been studied in the last years, the solutions adopted ranging from domain specific to general modeling languages [Otter and Cellier, 1995]. The philosophies and the methodologies used in modeling languages are diversified. However, two main categories are identified: those using traditional programming techniques and those using object-oriented methodologies. Nowadays, the focus is centered toward object-oriented languages mainly due to its simplicity in reusing, expanding and adapting models. Confirming this idea is the work done by [Beater, 1998], refering that the largest time consuming step in system modeling can be speed up by using modern object-oriented simulation languages and component libraries.

The integration of models from different domains is a complex and time consuming task because, although there are powerful libraries, they are generally based on different modeling languages, almost invariably, not compatible. To minimize this situation there is a strong effort of a working group including simulation tool builders, computer scientists and users from different domains in order to build a unified object-oriented language for physical systems’ modeling. This language, named Modelica, is intended for modeling virtually any application domain (electrical circuits, multi-body systems, hydraulics, thermodynamic systems, chemical systems, etc.), and possible incorporation of several formalisms for behavior description (ODE, DAE, bond graphs, finite state automata, petri nets or statecharts).

It is believed that the association of an object oriented modeling language, for model structure description (such as Modelica), with a graphical formalism suitable for dynamic behavior description of hybrid systems (such as the formalism of Statecharts) can be an interesting and useful approach for electro-hydraulic systems’ modeling.

3. Hybrid Statecharts and Modelica language

A brief review on the formalism of Statecharts

The formalism of Statecharts [Harel, 1987] is intended to describe the dynamic behavior of complex reactive systems. It is viewed as an extension of the finite-state-machine (FSM) formalism with the add-ons of hierarchy, parallelism and broadcast communication.

Hierarchy is a well accepted approach for dealing with complexity. In fact, it helps the human abstraction process and is generally accepted as a basic feature of modern computer software. In Statecharts, hierarchy is used to group sets of states together. This hierarchical structuring of states allows high level description and step-wise development. The designer can start by an high level description of the model and then by the refinement of states with AND/OR decomposition. This hierarchical organization encourage “zoom” capabilities for moving easily back and forth between different levels of abstraction.

Concurrency inside a statechart can be described through AND states, allowing modeling of concurrent activities (parallelism) in the same model through orthogonal states. All these orthogonal states are activated when an AND state is entered and deactivated when an AND state is exited.

Figure 1 shows the high level description of a system with states A and B. These states can be refined through state decomposition. State A will be a compound OR state with substrates A1, A2 and A3. State B will be a compound AND state with substrates B1 and B2. This
process can proceed until low level description is achieved.

When orthogonal components are not truly independent, communications between orthogonal components should be specified. This communication is achieved by associating an action with a transition. This action is assumed to be broadcasted so every system component in the statechart will recognize the message. This broadcast communication mechanism allows that, when one part generates an event (attaching an action to a transition), all the other parts sense it, acting in response if it is so specified. Looking at the example of figure 2, state A generates an event when transition f is fired. This event is sensed if state G is active although state G belongs to another orthogonal state.

Another enhancement over the FMS formalism is the possible association of an event action with a transition, when transition is taken, or with a state, when the state is entered or exited. Continuous activities can also be associated with a state for modeling continuous behavior when the state is active. These actions and activities make easier the modeling of hybrid systems: actions capture the discrete features of the system while activities describe the continuous part of the system.

Modelica

Modelica is a new language for physical modeling that is being developed as an international effort with the main objective of making easy the exchange of models and model libraries. The language is built on non-causal modeling with algebraic and differential equations, and uses object-oriented constructs to facilitate model reusing, through hierarchical modeling, encapsulation, and inheritance.

Models and submodels are declared as classes with connection interfaces called connectors. This connection possibility allows the use of model libraries to compose complex models with the drag and drop, and connection drawing facilities of modern graphical editors.

With Modelica, hybrid systems modeling is supported via mixed continuous/discrete systems of equations. Discontinuous models can be handled with if-then-else expressions, allowing modeling of phenomena with different expressions in different operating regions. Models with different complexities can be supported with conditional equations, making changes on behavior’s descriptions by just setting a parameter. Discrete event and discrete time models are supported by when statements. The equations in a when clause are conditionally activated at event instants where the when condition becomes true.

The statecharts library in Modelica

A small library to implement the hybrid statecharts formalism in Modelica language was already developed [Ferreira et al., 1999]. Two statecharts implementation levels were considered: statecharts library models and component models. Statecharts library models are responsible for capturing events related to the firing of transitions and with activation and deactivation of states that must be performed when transitions are taken. Library models have also to code the activation or deactivation of the substates, in order to implement what is expected with OR states and AND states. The component models create the statechart with the basic models provided by the statecharts library, making the state-transition-state connections, defining the transition events or describing the continuous activities within states. The approach followed to generate the code for statechart implementation is to consider a statechart as a model in Modelica. This model will be composed by
states and transitions. Modelica models were developed for the basic elements of a statechart. The final model of a statechart is a set of Modelica models connected by state-transition-state connectors.

The equation based modeling of the Modelica language, along with the connector constructions, proves its efficiency by passing activation/deactivation messages instantaneously through nested states. Also, the broadcast communication mechanisms of statecharts can easily be fulfilled, just by setting the value of a variable; this is automatically transmitted to all the statechart components because, in fact, the statechart is implemented with differential algebraic equations (DAE) that are evaluated concurrently.

**Behavior inheritance of the Statecharts library**

The main guideline followed by Harel [Harel and Gery, 1997], concerning the statecharts behavior inheritance, is to base the two statecharts on the same underlying state/transition topology. Thus, B inherits all A’s states and transitions. Although these cannot be removed, certain changes could be allowed. States could be modified in three ways: decompose a basic state by OR (into substates) or by AND (into orthogonal components) decompositions; add substates to an OR state; add orthogonal components to any state.

This last way is the most important because it is used to enrich A’s behavior capabilities. Transitions can also be added to the statechart, and some modifications could be allowed in the inherited ones. For example, if the transition is labelled by \texttt{event[guard]action}, changes could be made in the trigger \texttt{event}, the \texttt{guard} or even in the \texttt{action} list. Although it is not possible to explicitly remove a transition, it can be done implicitly by making its \texttt{guard} false.

All these features were implemented in the statecharts Modelica library by means of boolean switch parameters associated to the status of each state or transition. That is, each state or transition can be inhibited, at compile time, by setting its enabling parameter to false.

**4. Modeling methodology**

The main directions of the methodology adopted, for modeling the dynamic behavior of modern hydraulic systems, are: the use of object oriented libraries of models for different physical domains; hierarchical description of the dynamic behavior; complex models are built through model interconnection; different simulation experiences are achieved by the use of different complexity levels for the models; possibility of refining or redefining behavior; graphical description of dynamic behaviors to enhance model understanding.

The concept behind the methodology is to consider a model, of a physical component, as a composition of two complementary perspectives: its structure and its behavior.

The structure models the static part of the model, its parameters, its connection terminals, etc. The use of an object oriented philosophy is assumed as the more appropriated for complex system structure modeling. The design of hierarchies of models, connected by inheritance mechanisms, is an important approach to develop reusable model’s libraries. Mechanisms for model interconnection could also make easier complex system’s modeling.

The dynamic part of the model, that is, its time evolution, is described by its behavior. This behavior depends on time, captured events or changing attributes deriving from model’s connection terminals. The behavior of an hydraulic model could be reactive when stimulated through its connections terminals. Thus, it is believed that a graphical formalism (such as Statecharts), suitable to describe reactive hybrid systems, is very useful for hierarchical description of the model’s dynamics.

The proposed methodology holds in the association of an object oriented modeling, equation based, language (Modelica) for model structure description, with a graphical formalism (Statecharts) suitable for dynamic behavior description of reactive hybrid systems. The implementation of this graphical formalism in the modeling language allows that the methodology can be supported by just one global language.

The association of the object oriented modeling language with the graphical formalism lead to two types of inheritance for component models. While the model structure can be inherited through the inheritance mechanisms of object oriented languages, this work proposes a way for behavior inheritance by using the hybrid statecharts formalism for behavior model description.

The organization of model’s knowledge is easy because the model’s behavior can be hierarchically described by nesting statecharts. In fact, the statecharts states can be decomposed until low level behavior description is achieved. An interesting feature of this approach is the possibility of inheriting and refining model’s behavior in a very understandable manner, and with well-defined rules; this allows the organization, for the same component, of models whose behaviors can
have different complexities suitable for different simulation experiments. For example, component models can be organized with a complexity order number; then, by just defining a certain degree of complexity, a complex model, can be composed by several connected component models, either for real time simulation or for off line simulation experiments. The user can always inhibit the behavior of models and define a new behavior, for a part or for the whole model, in a very elegant way.

The relief valve example

Consider the example of a hydraulic component, relief valve (fig. 3), that limits the pressure in a hydraulic system. It is a closed loop system but, it is usually modelled by its (static) input/output characteristic [Beater, 1998]. Consider, in a first approach, that the valve has the behavior described by the graphic on the left side of figure 4.

Taking the simple relief valve model its behavior could be described considering that the valve has two possible stable states, and has hysteresis when changes from one state to another. When the pressure difference \( dp \) is bigger than the pressure required to open the valve \( \text{pressureOpen} \) the valve will be totally opened \( q = (\text{dp} - \text{pressureClose}) \cdot g_{\text{Open}} \); when the pressure difference is smaller than the close pressure \( \text{pressureClose} \) the valve will be totally closed \( q = \text{dp} \cdot g_{\text{Leak}} \).

Using the formalism of statecharts, the dynamic behavior of the valve (fig. 5) can be represented by an OR statechart (\text{ReliefValve}) that contains two substates (\text{Close} and \text{Open}) and two state transitions (\text{tOpen} and \text{tClose}). The behavior is the following: initially the valve is closed (the substate \text{Close} is the default substate). The continuous behavior of the relief valve is described by its during activity \( q = \text{dp} \cdot g_{\text{Leak}} \) that is enabled while the valve is in its \text{Close} state and is not exiting. If the valve is totally closed and its pressure difference \( \text{dp} \) exceed \\( \text{pressureOpen} \) then a transition \( \text{tOpen} \) takes place for the state \text{Open}. The during activity of state \text{Open} will be evaluated in all the simulation steps while the state \text{Open} is active, that is, while transition \text{tClose} do not take place. This transition takes place when pressure \( \text{dp} \) have a value below the pressure difference defined by the parameter \text{pressureClose}.

Refining of the relief valve

To improve the relief valve model, the characteristics represented at the right of figure 4 can be used. One of the possibilities is carried out by the statechart associated with the \text{ReliefValveExt} model of figure 5. The statechart shows that the states \text{Close} and \text{Open} and transition \text{tOpen} were redefined. The transition \text{tClose} is the only one that is inherited without changes from the ancestor model (\text{ReliefValve}).

It should be noted that the state \text{Open} was refined, being now an OR state with \text{PartialOpen} and \text{TotalOpen} substates. The model will inherit all the structure and, concerning the behavior, all the equations that were not
redefined. For instance, the equation that defines the trigger event for the transition \( t_{Close} \) (\( t_{Close}.event = dp < pressureClose \)) will be inherited from the model \( ReliefValve \), while the trigger event equation for transition \( t_{Open} \) must be redefined. A subset of the Modelica code for the relief valve model’s hierarchy is presented below. These models use the Statecharts Modelica library described in [Ferreira et al., 1999].

\[
\text{model HydOilProp “Oil properties”}
\]
\[
\text{parameter Real } KVisc = 46e-6; \quad \text{m}^2/\text{s}; \text{kinematic viscosity}
\]
\[
\text{parameter Real } \rho = 865; \quad \text{kg/m}^3; \text{mass density}
\]
\[
\text{parameter Real bulkModulus = 1e9; } \quad \text{Pa; bulk modulus}
\]
\[
\text{end HydOilProp;}
\]

\[
\text{connector HydConnector “Hydraulic connector”}
\]
\[
\text{Real } p; \quad // \text{Pa; fluid pressure}
\]
\[
\text{flow Real } q; \quad // \text{m}^3/\text{s}; \text{fluid flow}
\]
\[
\text{end HydConnector;}
\]

\[
\text{model TwoPortHydComp}
\]
\[
\text{extends BasicOil;}
\]
\[
\text{HydConnector HydA; // hydraulic connector A}
\]
\[
\text{HydConnector HydB; // hydraulic connector B}
\]
\[
\text{Real } \Delta p; \quad // \text{Pa, pressure difference}
\]
\[
\text{Real } q; \quad // \text{m}^3/\text{s}; \text{flow through component}
\]
\[
\text{equation}
\]
\[
\Delta p = \text{HydA.p - HydB.p;}
\]
\[
\ldots
\]
\[
\text{end TwoPortHydComp;}
\]

\[
\text{model ReliefValve}
\]
\[
\text{extends TwoPortHydComp;}
\]
\[
\text{parameter Real pressureOpen = 55e5; } // \text{Pa}
\]
\[
\ldots
\]
\[
\text{RootStateS Root;}
\]
\[
\text{StateS Close (defaultState = true);}
\]
\[
\text{StateS Open;}
\]
\[
\text{TransitionS tOpen;}
\]
\[
\text{TransitionS tClose;}
\]
\[
\text{equation}
\]
\[
t_{Open}.event = \text{event } (\Delta p > \text{pressureOpen});
\]
\[
t_{Close}.event = \text{event } (\Delta p < \text{pressureOpen});
\]
\[
q = \text{if Open.active } \text{ then } (\Delta p - \text{pressureClose}) * g_{Open}
\]
\[
\text{else } \Delta p * g_{Leak;}
\]
\[
\text{end ReliefValve;}
\]

\[
\text{model ReliefValveExt}
\]
\[
\text{extends ReliefValve;}
\]
\[
\ldots
\]
\[
\text{StateS PartialOpen (defaultState = true);}
\]
\[
\text{StateS TotalOpen;}
\]
\[
\text{TransitionS tPOpen;}
\]
\[
\text{TransitionS tTOpen;}
\]
\[
\text{equation}
\]
\[
t_{Open}.event = \text{event } (\Delta p > \text{pressureClose});
\]
\[
t_{POpen}.event = \text{event } (\Delta p > \text{pressureOpen});
\]
\[
t_{TOpen}.event = \text{event } (\Delta p < \text{pressureOpen});
\]
\[
q = \text{if Closed.active } \text{ then } \Delta p * g_{Leak}
\]
\[
\text{else if PartialOpen.active } \text{ then } (\Delta p - \text{pressureClosed})^2 * \frac{g_{Open}}{(\text{pressureOpen} - \text{pressureClose})} + \Delta p * g_{Leak}
\]
\[
\text{else } (\Delta p - \text{pressureClosed}) * g_{Open} + \Delta p * g_{Leak;}
\]
\[
\text{end ReliefValveExt;}
\]

The dynamic behavior of instances of \( ReliefValve \) and \( ReliefValveExt \) models (\( r_{Valve} \) and \( r_{ValveExt} \), respectively) is shown in the following figure, when using the absolute value of a sinusoidal signal as the pressure difference \( \Delta p \). This model was tested with the Dymola package [Dymola] that supports the Modelica language.

![Fig. 6 – Behavior of ReliefValve and ReliefValveExt models](image)

This example shows how simple could be the refining of behavior of, for example, model libraries of hydraulic components, if using the present methodology proposal.

5. Hardware-in-the-loop simulation of hydraulic systems

Hardware-in-the-loop simulation refers to a technology where some of the components of a pure simulation are replaced with actual hardware. This type of procedure is useful, for example, to test a controller which, instead of being connected to the real equipment under control, is connected to a real time simulator. The controller must “think” that it is working with the real system and so the accuracy of the simulation and its electrical interfacing to the controller must be adequate [Maclay, 1997]. This technology provides a way for testing control systems over the full range of operating conditions, including failure modes. Testing a control
system prior to its use in a real plant can reduce the cost and the development cycle of the overall system. Hardware-in-the-loop simulation has been used, with success, in the aerospace industry and is now emerging as a technique for testing electronic control units. This procedure has been applied to solve some specific problems but is seldom used as a platform to test the real time behavior of hardware components.

![Fig. 7 – Outline of the block diagrams of hardware and software components](image)

The main purpose of the present methodology is to give a well defined support for the model libraries of electro-hydraulic components, to be used in hardware-in-the-loop simulation experiments. The global performance is related to the model’s complexity, thus, for different type of simulations, different model’s behavior shall be used. When using real time simulation, the model’s complexity shall take into account the dedicated real time hardware that will “run” the codified model. With the proposed methodology, the real time simulation experiments can be done with the same model topology as the off-line simulations, because the time required for simulation can be defined (before simulation) by selecting the appropriate degree of complexity.

In order to use this methodology for hardware-in-the-loop simulation of electro-hydraulic systems, libraries of models with different complexities should be developed for the different physical domains involved in electro-hydraulic systems. This task deals with refining, reusing or developing new basic models for the different physical domains involved.

The main use of hardware-in-the-loop simulation in hydraulic systems is the testing of control algorithms and the development of new control schemes. Usually this implies a real controller, operating over a real time simulated hydraulic plant.

The objective of future work is to be able to modify the complexity of a composed hydraulic model, just by setting parameters; that way a model can be compiled with the complexity adjusted to the hardware platform.

Figure 7 presents an outline of the general block diagram for the modeling methodology testbench. This platform will be used to emulate electro-hydraulic systems in order to test real controllers and algorithms.

6. Conclusions

The present paper proposes a methodology to organize model libraries of electro-hydraulic components, in order to easily manage complex models for hardware-in-the-loop simulation experiments.

The concept behind the methodology is the following: the model of a physical component is composed by two complementary perspectives: its structure and its behavior. For that, the methodology holds on the association of an object-oriented language, Modelica, to model the structure, and the hybrid Statecharts formalism to describe the dynamic behavior of the model.

Making use of the Statecharts inheritance rules, this work proposes a well defined way to refine or redefine the behavior of hydraulic systems. This way, different simulation experiments can be achieved by choosing the appropriate behavior for each model.

7. References


