Modeling of the ARGESIM "Crane and Embedded Controller" System using the DEVSLib Modelica library

Victorino Sanz [*,⋆] **François E. Cellier** [**] **Alfonso Urquia** [*] **Sebastian Dormido** [*]

[*] *Dpto. Informática y Automática, ETSI Informática, UNED, Juan del Rosal 16, 28040, Madrid, Spain (e-mail: {vsanz,aurquia,sdormido}@dia.uned.es).*
[**] *Institute of Computational Science, Dept. of Computer Science, ETH Zurich, CH-8092 Zurich, Switzerland (e-mail: francois.cellier@inf.ethz.ch).*

Abstract:
DEVSLib is a free Modelica library, developed by the authors, that supports the Parallel DEVS formalism. The library is mainly designed to model discrete-event systems. It also includes interfaces to communicate the DEVSLib models with the rest of the Modelica libraries. Thus, the library can be used in the development of multi-domain and multi-formalism hybrid models. This manuscript discusses the modeling of the system "Crane and Embedded Controller," proposed by ARGESIM, using Modelica and DEVSLib. The crane system is composed of a car that moves along a rail and a load connected to the car by a cable. A discrete controller controls the position of the car and its movement. The crane system is implemented in Modelica as a continuous-time model and the discrete controller is constructed using DEVSLib. The communication between the continuous-time and the discrete-event parts is performed using the DEVSLib interfaces. DEVSLib is freely available for download at `http://www.euclides.dia.uned.es`.

Keywords: Object-oriented modeling, hybrid systems, Modelica, DEVS.

1. INTRODUCTION

This manuscript discusses the implementation of the ARGESIM comparison "Crane and Embedded Controller". ARGESIM is a non profit working group providing the infrastructure and administration for dissemination of information on M&S in Europe (ARGESIM, 2009).

The system consists in a crane controlled by a discrete controller. The crane is composed of a car that moves along a horizontal rail and a load connected to the car by a cable. This system was proposed by ARGESIM as a comparison for different tools that support hybrid modeling.

Various authors implemented this system using different tools or languages. Some implementations were based on the original definition of the system (Scheikl et al., 2002). These implementations were developed using Matlab (Scheikl, 2001; Schachinger, 2002; Wöckl and Breitenecker, 2003; Weidinger and Breitenecker, 2003), Anylogic (Garifullin, 2003) and VHDL-AMS (Wang and Kazmierski, 2005). Another implementation, based on the revised definition of the system (Schiftner et al., 2006), uses Modelica/Dymola (Schiftner, 2006).

The original and revised definitions of the system differ in the design of the controller. In the original definition, the controller receives as inputs the angle of the load and the position of the car. In the revised definition, only the latter variable is received. Also, the equations of the controller and its parameters are improved in the revised definition.

DEVSLib is a free Modelica library that supports the Parallel DEVS formalism (Chow, 1996; Zeigler et al., 2000). Additionally, it includes interface models to transform DEVS events into continuous-time signals and vice-versa. Thus, DEVSLib is compatible with the other Modelica libraries. The goal of this paper is to demonstrate, with a case study, the hybrid modeling capabilities of the DEVSLib library together with other Modelica libraries.

The implementation described in this manuscript follows the revised definition of the system. The crane system has been described as a continuous-time model using the Modelica language. The controller has been modeled in part using the DEVSLib library and in part using the Modelica Standard Library (MSL). Both parts are interconnected using the DEVSLib interface models.

The DEVSLib library has been developed by the authors of this manuscript and is freely available on the web (EuclidesWebSite, 2009). The library and the work discussed in this paper have been developed using the Dymola simulation environment (DynasimAB, 2006).

The document is organized as follows. The DEVSLib library is briefly described in the next section including its

⋆ This work has been supported by the Spanish CICYT under DPI2007-61068 grant.

interfaces for hybrid system modeling. The crane system and its implementation in Modelica as a continuous-time model are described in Section 3. The discrete controller and its components, including their specification and implementation, are described in Section 4. The results obtained from the developed model, after simulating the tasks described in the definition, are presented in Section 5. The paper ends with some conclusions.

2. DEVSLIB MODELICA LIBRARY

The general architecture of DEVSLib is shown in Fig. 1. The basic model components that can be defined using DEVSLib are called "atomic" models. An atomic model is specified in the Parallel DEVS formalism with the tuple $M = (X, S, Y, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$, where X is the set of input ports, Y is the set of output ports, S is the set of states, δ_{ext}, δ_{int} and δ_{con} are the transition functions, λ is the output function and ta is the time advance function.

The behavior of an atomic model is guided by two types of events: external and internal. The external events are received through the input ports and trigger external transitions, managed by the external transition function (δ_{ext}). An internal event occurs when no external events are received for a given period of time, calculated using the time advance function (ta). This triggers the internal transition (δ_{int}). The confluent transition function (δ_{con}) is used to manage confluent events, which occur when an external and an internal event are received simultaneously. Before executing the internal transition function, the model can generate output events using the output function (λ).

Atomic models in DEVSLib (see model "atomicDraft" in Fig. 1) can be described similarly to their formal specification (M). The interfaces of the model are defined including the required input and output ports. The state is represented as the Modelica record "st" and can be initialized defining the "initst" function. The transition, output, and

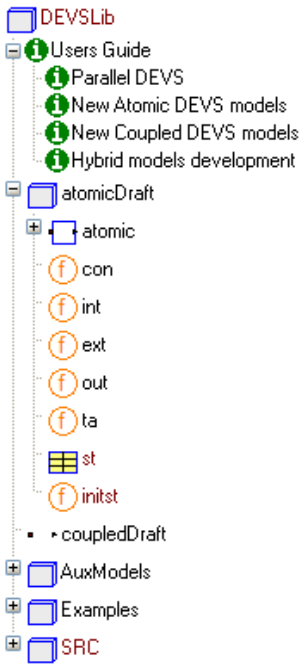


Fig. 1. DEVSLib library architecture

time advance functions can be defined be means of the "int," "ext," "con," "out," and "ta" functions.

External events transport information between models. The transmitted information is called "message". Messages are created by the output function and are deleted upon reception by the external transition function like in a message passing mechanism.

The model communication mechanism in Modelica is based in the definition of ports, called "connectors," and connections between ports, using "connect-equations". Variables defined in two connected connectors are either set equal, or are summed up with the sum being set equal to zero.

The Modelica model communication and the DEVS message passing mechanisms are conceptually different. The former relates values of variables, while the latter transports information between models. Several approaches were studied and developed in order to implement a suitable message passing mechanism in Modelica.

A direct implementation of a message passing mechanism in DEVSLib using Modelica connectors was studied. However, connectors do not allow the simultaneous reception of messages, because their variables cannot be assigned with several values at the same time. Also, Modelica does not allow a variable number of objects in a model, so the message transmission cannot be directly implemented.

Other approaches for developing the message passing mechanism in DEVSLib, based in an intermediate storage of the transmitted messages, were studied and implemented. The first approach was to use a text file to store the messages, so the sender writes the message to the file and notifies this to the receiver, that subsequently reads it. This approach allows simultaneous reception of messages, because several messages can be written to the file, but its performance and versatility are poor. The other approach substitutes the text files by dynamic memory space. This increases the performance and the versatility of the mechanism, allowing to manage different types of messages without redefining the message management operations.

The dynamic memory approach for message passing is the mechanism implemented in DEVSLib. This approach is combined with the standard Modelica connectors to provide a transparent communication mechanism to the user. DEVSLib models are topologically connected using standard Modelica connectors and connect-equations.

DEVSLib models can be aggregated and connected to compose "coupled" models. Coupled model components can be either atomic or other coupled models, thus the coupled models can be arranged hierarchically. A coupled model is specified in the Parallel DEVS formalism with the tuple $M = (X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC)$, where X is the set of input ports, Y is the set of output ports, D is the set of component names, M_d is the set of DEVS components, EIC is the set of connections between input ports and components, EOC is the set of connections between components and output ports and IC is the set of internal connections between components.

The development of coupled models with DEVSLib also follows its formal specification. Using the object-oriented

modeling capabilities of Modelica, coupled models are constructed connecting previously developed components and including the required input/output ports.

2.1 DEVSLib Interfaces to Other Modelica Libraries

DEVSLib includes models to translate the content of the messages into continuous-time signals, and vice-versa. There are two mechanisms used in the continuous-to-discrete translation: the cross-functions and the quantization. The former translates the value of a continuous-time signal into a message every time the signal crosses a given threshold in one direction (upwards or downwards). The models "crossUP" and "crossDOWN" implement this behavior in DEVSLib. The quantization mechanism is implemented by the "quantizer" model. This model generates a message every time the value of the continuous-time signal changes by a predefined quantum, similarly to the behavior of the QSS first-order method (Kofman, 2004).

On the other hand, the discrete-to-continuous translation is performed generating a piecewise-constant signal whose value is the one of the last message received. The model "DICO" (DIscrete-to-COntinuous) implements this behavior in DEVSLib.

These interface models are implemented to manage the standard DEVSLib message type. However, the message type in DEVSLib can be redefined by the user and the interface models adapted to the new message type.

3. CRANE SYSTEM MODEL

The crane system is composed of the car, the cable, and the load (see Fig 2). The discrete controller controls the position of the car to reach the desired position, specified by the user. A detailed description of the system is given in (Schiftner et al., 2006).

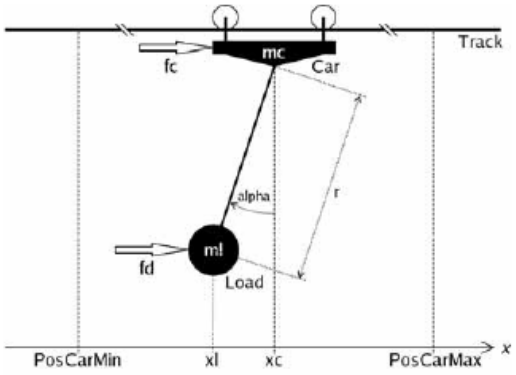


Fig. 2. Scheme of the crane system (Schiftner et al., 2006)

The car moves along the track in accordance with a force f_c, provided by a motor. The force of the motor is calculated using the following first-order ODE (Ordinary Differential Equation) (Schiftner et al., 2006):

$$\dot{f}_c = -4(f_c - f_c^{desired}) \tag{1}$$

Where $f_c^{desired}$ is the signal generated by the discrete controller. The car can also be stopped using a brake, whose activation conditions will be detailed below. The movement of the car is restricted to the values $PosCarMax$ and $PosCarMin$. The load hangs from the car by a cable. Similarly to the car, the load is influenced by a force f_d, that represents some disturbances.

Table 1. Model variables

Symbol	Description	Unit
α	angle of the cable	rad
f_c	motor force	N
f_d	load disturbances	N
x_c	position of the car	m
x_l	position of the load	m

Table 2. Model parameters

Symbol	Description	Value
d_c	friction coefficient of the car	0.5 kg/s
d_c^{brake}	friction coefficient of the car with activated brake	10^5 kg/s
d_l	friction coefficient of the load	0.01 kg/s
g	gravity	9.81 m/s^2
m_c	mass of the car	10 kg
m_l	mass of the load	100 kg
$PosCarMax$	maximum position of the car	5 m
$PosCarMin$	minimum position of the car	-5 m
r	length of the cable	5 m

Three sensors are used to observe the state of the system:

(1) The position of the car (named $PosCar$).
(2) The maximum position limit (named $SwPosCarMax$, is activated when $PosCar > PosCarMax$).
(3) The minimum position limit (named $SwPosCarMin$, is activated when $PosCar < PosCarMin$).

If either the $SwPosCarMax$ or the $SwPosCarMin$ sensor is active, the system enters $EmergencyMode$, which causes an emergency stop.

The equations (Schiftner et al., 2006) that describe the dynamics of the crane system are the following (variables and parameters are detailed in Tables 1 and 2):

$$\begin{aligned}\ddot{x_c}\left[m_c + m_l\sin^2(\alpha)\right] = &-d_c\dot{x_c} + f_c + f_d\sin^2(\alpha)\\ &+m_l\sin(\alpha)\left[r\dot{\alpha}^2 + g\cos(\alpha)\right] - d_l\dot{x_c}\sin^2(\alpha)\end{aligned} \tag{2a}$$

$$\begin{aligned}r^2\ddot{\alpha}\left[m_c + m_l\sin^2(\alpha)\right] = &\left[f_d\frac{m_c}{m_l} - f_c + d_c\dot{x_c}\right]r\cos(\alpha)\\ &-\left[g(m_c+m_l) + m_l r\dot{\alpha}^2\cos(\alpha)\right]r\sin(\alpha)\\ &-d_l\left[\frac{m_c}{m_l}(\dot{x_c}r\cos(\alpha) + r^2\dot{\alpha}) + r^2\dot{\alpha}\sin^2(\alpha)\right]\end{aligned} \tag{2b}$$

$$x_l = x_c + r\sin(\alpha) \tag{2c}$$

Equations (2a), (2b), and (2c) can be linearized (Föllinger, 1985) to obtain the following linear model, in order to simplify the model and allow a comparison with the nonliner model:

$$\ddot{x_c} = \frac{f_c}{m_c} + g\frac{m_l}{m_c}\alpha - \frac{d_c}{m_c}\dot{x_c} \tag{3a}$$

$$r\ddot{\alpha} = -g(1+\frac{m_l}{m_c})\alpha + (\frac{d_c}{m_c} - \frac{d_l}{m_l})\dot{x_c} - r\frac{d_l}{m_l}\dot{\alpha} - \frac{f_c}{m_c} + \frac{f_d}{m_l} \tag{3b}$$

$$x_l = x_c + r\alpha \tag{3c}$$

The model equations have been directly programmed in Modelica using equations (3a),(3b), and (3c) for the linear model, and (2a),(2b), and (2c) for the non-linear model. Both cases include (1), that models the motor. Dymola can internally handle the implicit equations of the non-linear model.

4. DISCRETE CONTROLLER MODEL

The complete system is shown in Fig. 3a. It corresponds to the discrete controller connected to the non-linear model of the crane system. The generators for the desired car positions and the load disturbances are also shown. The non-linear model can be substituted by the linear model. The controller, generators, and connections are compatible in both cases.

The controller is implemented as a cycle-based controller (Schiftner et al., 2006). It is composed of three parts: the state-space observer, the regulator, and the diagnosis module. The state-space observer calculates five "fictitious" states (q), and the regulator generates a control signal based on the observed states. Additionally, the diagnosis module manages the conditions for the *EmergencyMode* and the activation of the brake. The structure of the implemented controller is shown in Fig. 3b

4.1 Position Controller

The state-space observer and the regulator have been implemented with DEVSLib, as an atomic DEVS model. This model corresponds to the "PositionController" in Fig. 3b, and its Parallel DEVS specification is the following:

$$M = (X_M, S, Y_M, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta)$$

where:

$X_M = \emptyset$
$S = \{\Re^5 \times \Re\}$
$Y_M = \Re$
$\delta_{int}(\mathbf{q}_n, u_n) = (\mathbf{q}_{n+1}, u_{n+1})$

$$\text{where } \begin{cases} \mathbf{q}_{n+1} = (\mathbf{M} - \mathbf{d}\mathbf{c}^T)\mathbf{q}_n + \mathbf{d}PosCar + \mathbf{b}f_c^{desired} \\ u_{n+1} = VPosDesired - \mathbf{h}^T\mathbf{q}_{n+1} \end{cases}$$

$\delta_{ext}(q, u, e, X) =$ nothing since $X_M = \emptyset$
$\delta_{con}(q, u, e, X) =$ nothing since $X_M = \emptyset$
$\lambda(q, u) = max(min(u, ForceMax), -ForceMax)$
$ta(q, u) = cycle$

M, **d**, **c**, and **b** are the parameters of the observer, and V and **h** are the parameters of the regulator. *PosCar* and *PosDesired* are continuous-time inputs to the δ_{int} function, similarly to the description of the internal transition function in the DEV&DESS formalism (Zeigler et al., 2000).

The "PositionController" executes an internal transition at each controller cycle. The internal transition function calculates the new state of the observer (q_{n+1}), and updates the control signal (u_{n+1}). The output function (λ) uses the parameter *ForceMax* to saturate the control signal and generate the output that will be sent to the crane system.

The implementation of the "PositionController" is directly extracted from its specification, translating the actions

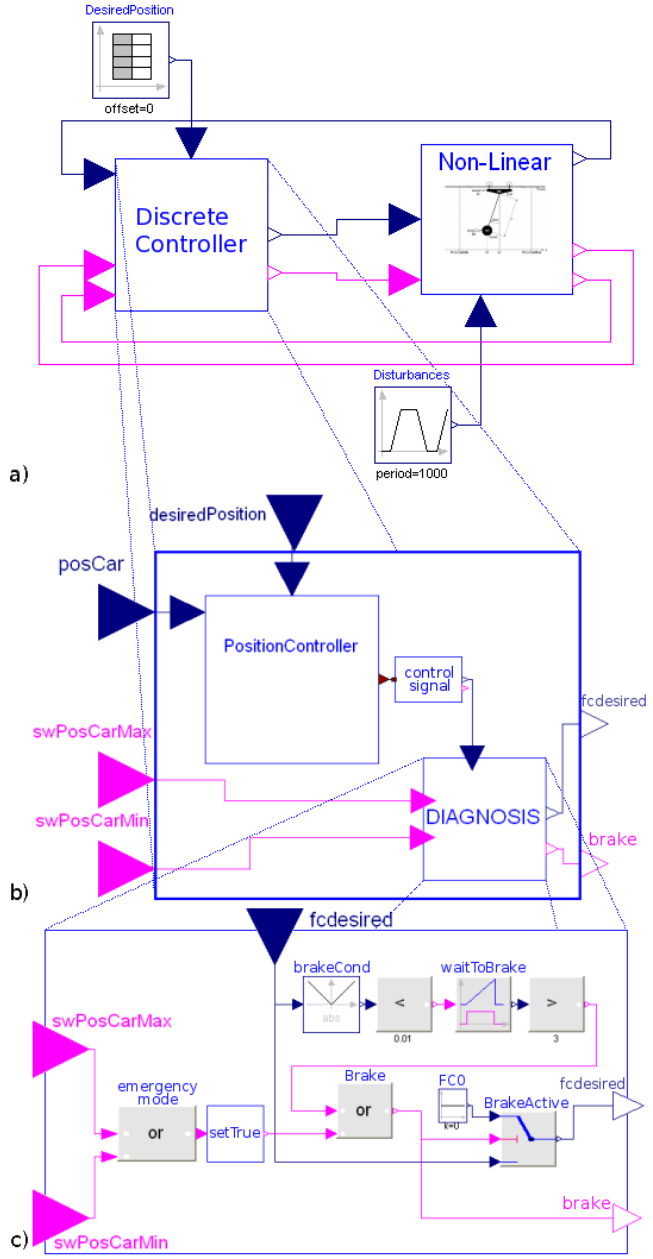


Fig. 3. "Crane and Embedded Controller" system: a) non-linear system with discrete controller; b) discrete controller implemented with DEVSLib and the MSL; and c) diagnosis module of the controller

performed by each transition function into Modelica functions. The output port (Y_M) is defined using a Modelica connector of the class "outPort", included in DEVSLib. The input parameters, *PosDesired* and *PosCar*, are defined using connectors from the MSL. Finally, the variables that define the state (S) has to be declared inside the Modelica record that represents the state of an atomic model in DEVSLib.

4.2 Interface Model

The output of the position controller is generated as a message. It contains the value of the control signal ($fcdesired$), which has to be translated into a continuous-time signal in order to be checked by the diagnosis module (see

Fig. 3b). The translation is performed by the "controlSignal" model, which is implemented using the mentioned DICO (DIscrete-to-COntinuous) interface model.

4.3 Diagnosis Module

The diagnosis module monitors the value of the control signal and the sensors $SwPosCarMax$ and $SwPosCarMin$. If any of the sensors becomes active, the controller enters $EmergencyMode$ and activates the brake. The brake is also activated when $|f_c^{desired}| < BrakeCondition$ for more than 3 seconds (where $BrakeCondition$ is a parameter of the controller).

This module (cf. Figs. 3b and 3c) has been implemented using components from the MSL (MSL, 2009). This demonstrates the compatibility between DEVSLib and previously developed Modelica Libraries (2009).

5. SIMULATION RESULTS AND DISCUSSION

Three tasks (A, B, and C), described in the definition of the system (Schiftner et al., 2006), have been performed in order to compare this implementation with previous results.

5.1 Task A

This task compares the implementation of the linear and the non-linear models without the controller and the brake. The input of the plant ($f_c^{desired}$) is set to 160 N during 15 s, and then to 0 N. The load disturbances initially start at 0 N ($f_d = 0$). At time = 4 s, $f_d = Dest$ for 3 s. The $Dest$ values are -750 N, -800 N, and -850 N. The system is simulated for each $Dest$ value during 2000 s, to reach the steady-state, and the position of the load in each model is compared. The results are shown in Table 3.

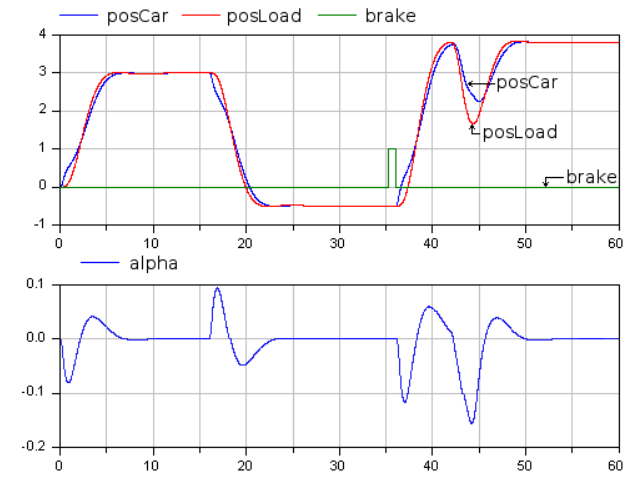
Table 3. Task A results

Dest	Linear	Non-Linear	Difference
-750 N	294.081 m	294.059 m	0.022 m
-800 N	-0.0048651 m	-0.0325238 m	0.0276587 m
-850 N	-294.091 m	-294.18 m	0.089 m

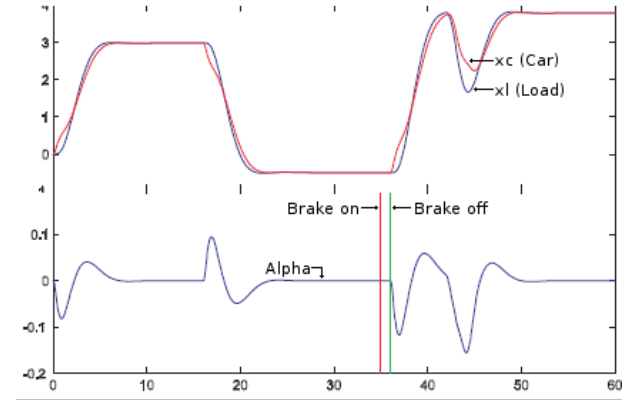
The results obtained are very similar to the ones reported in Schiftner (2006). The differences between the two models are -0.034 m, 0.013 m, and -0.016 m. The slight differences between the two implementations are explained by the different implementation of the non-linear model – using the MultiBody library (Otter et al., 2003) instead of plain equations. The use of the DEVS formalism to describe the discrete controller facilitates its understanding and development, in comparison with its description in plain Modelica code.

5.2 Task B

The next task describes how the non-linear model and the discrete controller work together. The desired positions for the car are 3 m at time 0 s, -0.5 m at time 16 s, and 3.8 m at time 36 s. The load disturbance is set to -200 N at time 42 s during 1 s. The results include the position of the car, the position of the load, the angle, and the activation of the brake over time. The system is simulated for 60 s. The results are shown in Fig. 4, comparing the implementation with DEVSLib with the one presented in Schiftner (2006).



a)



b)

Fig. 4. Task B results in: a) DEVSLib; and b) Schiftner (2006)

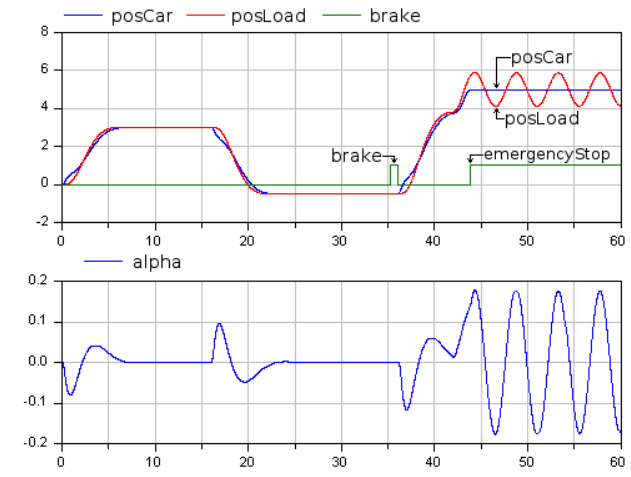
5.3 Task C

The last task evaluates the response of the system in case of an emergency stop. The scenario is the same as in the task B, but the value of the load disturbance is 200 N instead of -200 N. The results shown also include the position of the car, the position of the load, the angle, and the state of the brake over the simulation time. The system is simulated for 60 s. The simulation results are shown in Fig. 5, also comparing the two implementations.

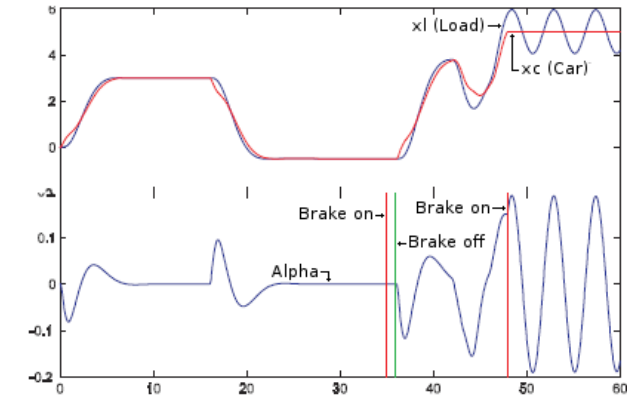
The emergency stop event is detected when the car reaches its maximum movement limit ($PosCarMax$). After that, the car stops and the load oscillates.

In this case there is a slight difference just before the emergency stop. This difference is due to the parameters of the experiment, because in the latter case the load disturbances are set to -200 N at time 42 s and to 200 N at time 46 s resulting in the observed delay of the emergency stop.

Similar comparisons can be performed with the results obtained in previous implementations (Wöckl and Breitenecker, 2003; Schachinger, 2002; Scheikl, 2001; Weidinger and Breitenecker, 2003; Wang and Kazmierski, 2005; Garifullin, 2003). Their results are equivalent to the ones



a)



b)

Fig. 5. Task C results in: a) DEVSLib; and b) Schiftner (2006)

presented in this manuscript. However, these previous implementations are based on the original definition of the model. Thus, their results are slightly different mainly due to the inclusion of the angle sensor as an additional input to the controller and the different design of the control.

6. CONCLUSIONS

The system described in the ARGESIM comparison "crane and embedded controller" has been implemented using Modelica and the DEVSLib library. The simulation results obtained with this implementation are equivalent to the ones obtained by previous implementations of the same system, using different tools. Other tools used to model this system describe the discrete behavior of the controller using formalisms like Finite State Automata, StateCharts, or Petri Nets.

The DEVSLib library allows to define Modelica models in accordance with the Parallel DEVS formalism. The DEVSLib library is compatible with other Modelica libraries. The implementation of models with DEVSLib is analogous to its DEVS specification using functions to describe its behavior.

REFERENCES

ARGESIM (2009). ARGE simulation news group website. http://www.argesim.org.

Chow, A.C.H. (1996). Parallel DEVS: a parallel, hierarchical, modular modeling formalism and its distributed simulator. *Transactions of the Society for Computer Simulation International*, 13(2), 55–67.

DynasimAB (2006). Dymola Dynamic Modeling Laboratory User's Manual. http://www.dymola.com/.

EuclidesWebSite (2009). Some free modeling and simulation resources, Dpto. Informática y Automática, UNED. http://www.euclides.dia.uned.es/.

Föllinger, O. (1985). *Regelungstechnik (5. Auflage)*. Hüthig.

Garifullin, M. (2003). An object-oriented hybrid approach to ARGESIM comparison 'C13 Crane and Embedded Control' with AnyLogic. *Simulation News Europe*, 37, 29.

Kofman, E. (2004). Discrete event simulation of hybrid systems. *SIAM Journal on Scientific Computing*, 25(5), 1771–1797.

Modelica Libraries (2009). Modelica free and comercial libraries. http://www.modelica.org/libraries.

MSL (2009). Modelica standard library. http://www.modelica.org/libraries/Modelica.

Otter, M., Elmqvist, H., and Mattsson, S.E. (2003). The new Modelica MultiBody library. In *Proceedings of the 3rd International Modelica Conference*, 311–330.

Schachinger, D. (2002). 'C13 Crane and Embedded Control' MATLAB hybrid approach. *Simulation News Europe*, 35/36, 83.

Scheikl, J. (2001). 'C13 Crane and Embedded Control' - MATLAB numerical simulation / event-oriented model. *Simulation News Europe*, 31, 31.

Scheikl, J., Breitenecker, F., and Bausch-Gall, I. (2002). Comparison C13 crane and embedded control – definition. *Simulation News Europe*, 35/36, 69 – 71.

Schiftner, A. (2006). A Modelica approach to ARGESIM comparison 'Crane and Embedded Control' (c13 rev.) using the simulator Dymola. *Simulation News Europe*, 16(1), 30.

Schiftner, A., Breitenecker, F., and Ecker, H. (2006). 'Crane and Embedded Control' – definition of an ARGESIM benchmark with implicit modelling, digital control and sensor action. revised definition – comparison 13 revised. *Simulation News Europe*, 16(1), 27 – 29.

Wang, L. and Kazmierski, T. (2005). VHDL-AMS - based hybrid approach to ARGESIM comparison 'C13 Crane and Embedded Control' with SystemVision. *Simulation News Europe*, 43, 30.

Weidinger, W. and Breitenecker, F. (2003). A classic CNS - solution to ARGESIM comparison 'C13 Crane and Embedded Control' using MATLAB. *Simulation News Europe*, 38/39, 59.

Wöckl, J. and Breitenecker, F. (2003). A directly programmed solution to ARGESIM comparison 'C13 Crane and Embedded Control' with MATLAB. *Simulation News Europe*, 37, 28.

Zeigler, B.P., Kim, T.G., and Praehofer, H. (2000). *Theory of Modeling and Simulation*. Academic Press, Inc., Orlando, FL, USA.