Interacting Modelica using a Named Pipe for Hardware-in-the-loop Simulation

Arno Ebner Anton Haumer Dragan Simic Franz Pirker
Arsenal Research
Giefinggasse 2, 1210 Vienna, Austria

phone: +43 50550 6663, fax: +43 50550 6595, e-mail: arno.ebner@arsenal.ac.at

Abstract

The paper presents a concept and an implementation of *Modelica* simulation interaction using the operating system inter-process communication method of the *Named Pipe*. The main aim of this presented work is to implement a hardware-in-the-loop simulation (HILS) environment based on *Dymola* which runs on a normal *Microsoft Windows* Personal Computer.

An energy storage test bench is connected by an analogue and digital data input/output card with the *Dymola* simulation computer. With this proposed system, particularly long-time simulations with sample rates up to 30 Hz can be executed very cost effective. Typical applications are simulations of drive cycles to test energy storage systems in electrified vehicles such as batteries or fuel cells. Other application examples are the verification of battery models, thermal management models or battery management system (*BMS*) models.

In this paper all methods used for implementation are described in detail. Especially the concept of inter-process communication and the concept for real-time and simulation time synchronization is discused.

An application example which uses the provided concept is also shown at in this paper. In this example a longitudinal simulation of a vehicle is presented. The startup phase of the internal combusting engine model and a short drive cycle in combination with a connected real battery is shown.

1 Introduction

The traditional approach for simulating technical or physical systems is to describe real systems in mathematical models. These systems are described with discrete equations or with continuous algebraic and differential equations. The simulation environment has a solver algorithm which generates a solution for the system model considering the initial values.

In some cases the user or other applications have to interact with the simulation, for example to:

- Start or interrupt the simulation
- Change parameters or variables during the simulation
- Communicate with other applications, for example with another simulation environment
- Exchange data with an input/output-card or a peripheral communication interface
- Build up a hardware-in-the-loop simulation environment

Hardware-in-the-loop (HIL) is the integration of real components and system models in a common simulation environment [1]. This means that some parts of a system, which should be tested, are virtual and other parts are real. HIL simulations are an important method for the development of mechatronic systems. An important advantage of HIL is that it allows function tests of mechatronic systems or components of such systems under simulated real conditions. Therefore HIL helps to save cost and time compared to conventional test runs on a real prototype.
There are three important consideations for the implementation of a hardware-in-the-loop simulation:

- The simulation of the dynamic system, in other words the mathematical or physical models must be processed in real-time.
- There must be synchronization between the time in the real world (the so called real-time) and the digital simulation-time of the simulation tool.
- The simulation tool must be able to communicate e.g. with others applications or an I/O communication interface.

The *Monitoring, Energy and Drives* division at *Arsenal Research* does research and development on components for Hybrid Electric Vehicles (HEV`s) and electrified auxiliaries for vehicles. For that they acquire know-how in the simulation of electric

drives, of energy storage systems and in vehicular simulations. A great deal of simulation models was built up in *Modelica* and simulated with *Dymola* [2], [3]. In order to verify and validate the implemented models and the developed system components, a connection to a hardware-in-the-loop simulation environment has to be implemented.
In that way in the provided paper an interaction of a *Modelica/Dymola* simulation with components outside of the simulation tool is shown. At this proposed HIL system the *Dymola* application runs on a standard Microsoft Windows Personal Computer. Actually two important processes run on the simulation PC: one process is the *Dymola* application and one process provides the input and output functionality for the peripheral card. Details on implementation for both tasks, first for the Dymola application using external functions and furthermore for the peripheral card application are given below.

This two processes communicate together using the inter-process communication (IPC) methods provided by the operating system. In the proposed case the tasks communicate via a so-called *Named Pipe* mechanism. Through the first-in first-out behavior of the *Named Pipe* communication method there is an implicit synchronization between the two processes. In this paper is described how to create and to open, how to write to and how to read from the *Named Pipe* and how to close it.

2 Inter-process Communication with Named Pipes

The IPC method that is used in this work is the so-called *Named Pipe* mechanism. This IPC method is available both on *UNIX* systems and on Microsoft Windows systems. The semantic and the function calls differ in the operating systems but the concepts are the same [4], [5]. Only the implementation on a *Microsoft Windows* operating system is shown in this paper.

Named Pipes are designed for the communication between the pipe server and one or more pipe clients. They have first-in first-out behavior and can be accessed like a file.

Stdio.h standard *C* library functions for file handling, such as *fopen()*, *fclose()*, *fread()* or *fwrite()* can be used to deal with *Named Pipes*. However, the usage of Microsoft Windows *Software Development Kit* (*SDK*) functions gives a more powerful access on the functionalities of *Named Pipes*. Specifically, using the *SDK* a pipe server calls the *CreateNamedPipe()* function to create an instance of a *Named Pipe*. The client calls the *CreateFile()* or *CallNamedPipe()* function to connect to an instance of the *Named Pipe*. *ReadFile()* and *WriteFile()* functions allow reading and writing to a specified *Named Pipe*.

In the presented work the *Dymola* process corresponds to the server process, which generates the pipe and waits for a connection with a client (for instance the I/O process). This process then executes the simulation steps, puts data in the communication pipe and gets data from the client process.

3 Implementations in Modelica and Time Synchronisation

The described mechanism of inter-process communication is implemented as an external function written in *ANSI/ISO C*. At *Arsenal Research* two main C functions for the communication of *Modelica* with other processes using *Named Pipes* are developed.

The *AllocateResources()* function creates and connects a *Named Pipe*. The function *PipeIO()* computes a string from the simulation variables array calculated by Dymola and writes this string in the *Named Pipe*. Then the function gets the data string from the *Named Pipe* and computes an array of *Real* variables for the *Dymola* simulation solvers.

In Figure 1 the flow chart of the functionalities of the server process is shown. Specifically, the function *AllocateResources()* and the function *PipeIO()* are described. For these functions a static library is build. This library is linked to the model using the function definitions. In *Dymola* a wrapper function to convert the external *C* function to a *Modelica* function is defined.

For a working HILS it is important, that the simulation time in the simulation application is synchronized with the real time. This synchronisation in *Modelica* is achieved by using the when clause in an algorithm statement:

```
...
when (time >= SimuNext) then
 (SimuNext, input) := PipeIO(time, output);
end when;
```

In this short code fragment `time` means the simulation time in *Dymola* and `SimuNext` means the real-time. `input` means an array of *Real* variables taken from external *C* code to *Modelica* and `output` means an array of *Real* variables taken from *Modelica* to external *C* code.

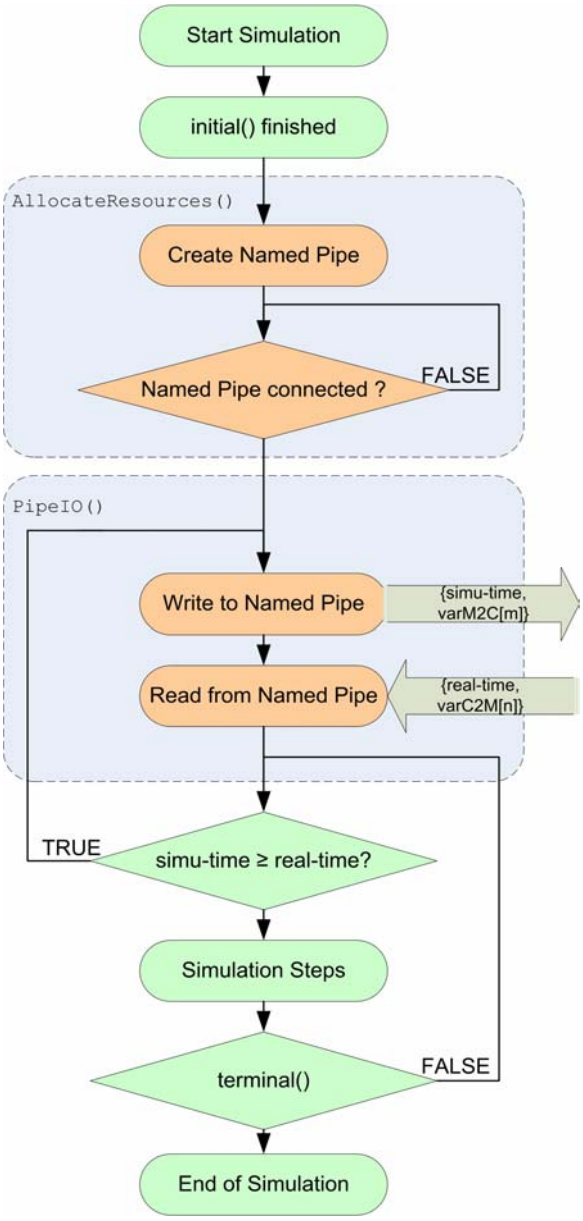

Figure 1: Flow chart of the server process, which runs as an external function call of Dymola.

The simulation process gets the real-time information from the client process via the *Named Pipe* IPC. As long as the real-time is greater than the simulation time in *Dymola* a new inter-process communication cycle get processed and the function *PipeIO()* gets called. As long as the real-time is smaller than the simulation time new simulation steps of the *Dymola* solver get executed.

In figure 2 the client process of *Named Pipe* IPC is shown. In this process the data in- and output functions and the calculation of real-time information for the simulation application using functions from the *ANSI/ISO C* library *time.h* are executed.

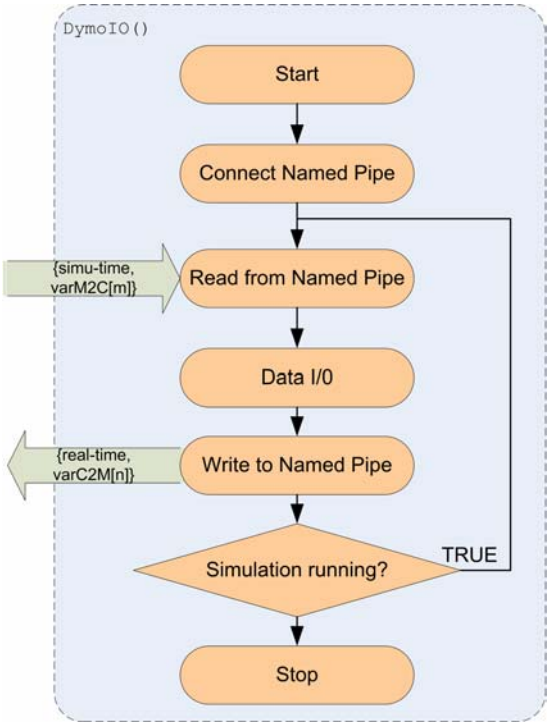

Figure 2: Flow chart of the client process.

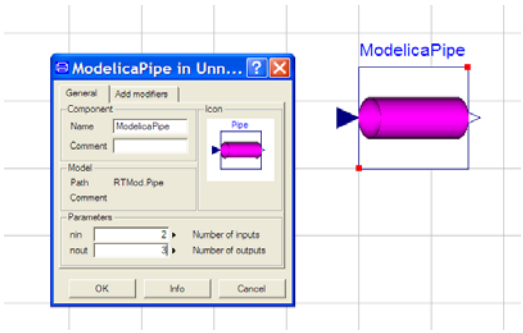

Figure 3: Icon and parameter settings of the Pipe block.

All inter-process and simulation time synchronization functionality are combined in the so called *Pipe* block. This block is shown in figure 3.

4 Application Example: Hardware-in-the-loop Energy Storage Test Bench

The HILS environment presented in this paper is a very cost efficient system which is based on a normal *Windows* PC platform.

The communication between the simulation PC and the energy storage test bench is realized by an analogue inputs and outputs card. Specifically, a commercial data acquisition (DAQ) Card from *National Instruments (NI)* was used for the implementation. By using the *DAQ* functions from *NI* the simulation

tool can communicate with the interface card [6]. The main components of the energy storage test bench are a stack of electronic power supplies and a stack of electronic loads.

The technical data of the energy storage test bench is:

- Voltage: 0V-600V
- Current: 600A charge and 750A discharge
- Peak Power: 96 kW

The target applications of this HIL-test bench are the development and the test of energy storage, thermal management and battery management models.

The concept diagram for this HILS-environment is shown in Figure 9. In this application example the vehicle model calculates the loads for the battery. During the simulation the instantaneous real battery conditions (Voltage, Current, and Temperatures) are considered by *Dymola*. The electric drives are modelled using the *SmartElectricDrives* library [7], [8].

The following simulation results show the first 30 seconds of the *New European Drive Cycle* (*NEDC)* of a conventional vehicle. After 2 second the internal combustion engine of the vehicle is started for 2 seconds, which is illustrated in Figure 4. Figure 5 shows the shaft speed of the ICE. It appears that the motor is running with idle speed after 4 seconds. During the starting process the starter motor has a defined current demand for the battery. In figure 6 this current demand is shown. At time t=4s the ICE is running in idle speed and at time t=10s the vehicle begins to drive following the NEDC. In these phases the alternator produces electrical power, which charges the battery of the vehicle.

With the proposed HIL interface the electronic load creates a real electric current drain for the real battery. The electronic power supply generates a real charging current for the real battery connected to the energy storage test bench. In figure 7 the measured current in the battery pins and in figure 8 the real voltage at the battery pins is shown.

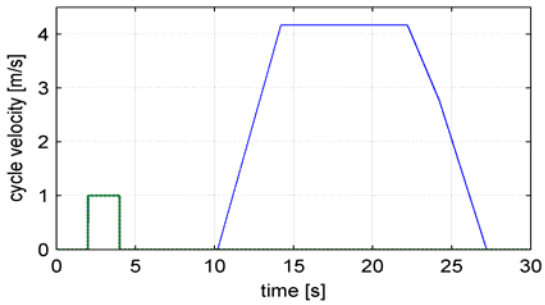

Figure 4: Starting the ICE after 2 seconds for 2 seconds and simulate first 30 seconds of the NEDC

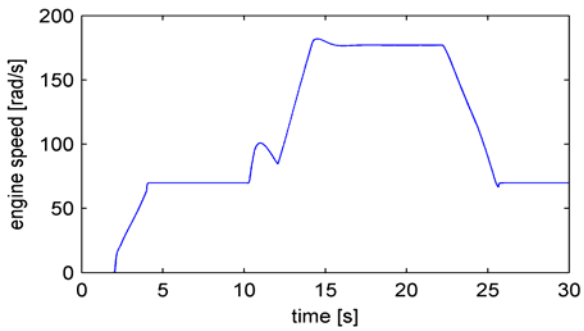

Figure 5: Simulation result, shaft speed of the ICE

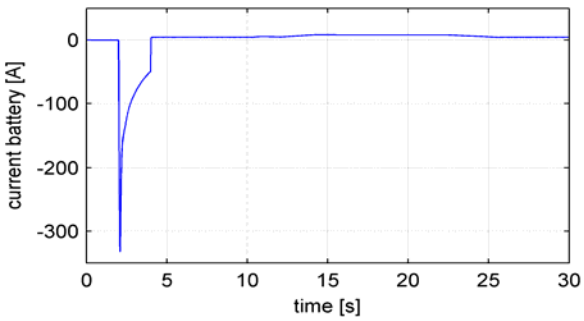

Figure 6: Reference current for battery – output of the HIL simulation

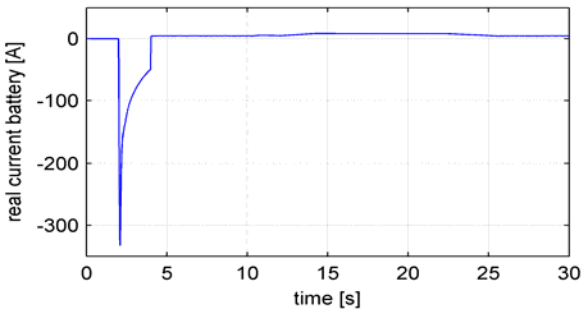

Figure 7: Real current in battery– input of the HIL simulation

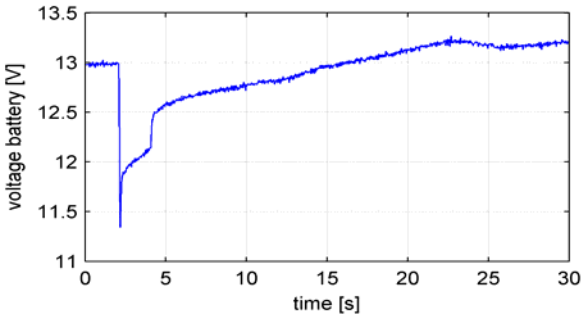

Figure 8: Real voltage at battery– input of the HIL simulation

This HIL simulation experiment was executed on a *MS Win32* PC with *Intel Pentium Mobile* processor with 1.8 GHz clock. In *Dymola* the *Dassl* integration method is used with a tolerance of 0.001. 25Hz Input/Output communication frequency was chosen.

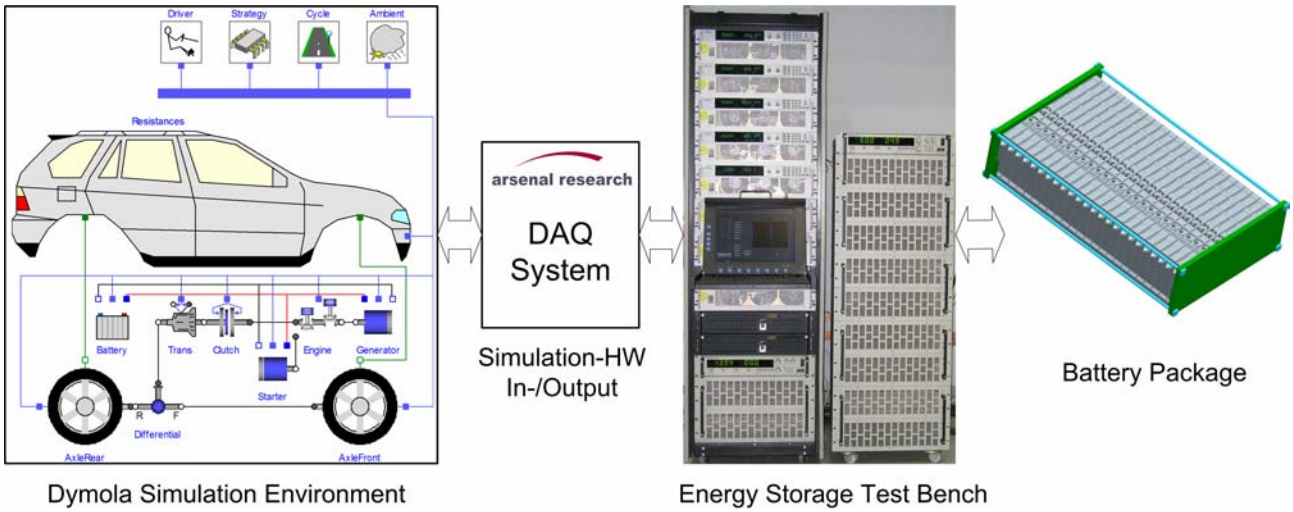

Figure 9: Application example, functional diagram of the HILS energy storage test bench

5 Conclusions

The *Modelica* hardware-in-the-loop simulation approach using Named Pipes inter-process communication methods was shown in this paper. This proposed system does not need an expensive "true" real-time platform. It bases on a normal *Windows* PC with a *Dymola* simulation environment and a communication interface card for inputs and outputs. Complex models can be simulated with an in-/output sample rate up to 30 Hz.

The HIL energy storage test bench which was proposed as application example in this paper is under operating conditions at *Arsenal Research* since the begin of 2006. With this HIL test platform particularly long-time simulations such as drive cycles test of energy storage systems or verifications of battery models, thermal and battery management models was done.

As future work *Arsenal Research* will implement the proposed inter-process communication algorithm also on a *Dymola* application which runs on a *Linux* PC.

Abbreviations

BMS	Battery Management System
DAQ	Data Acquisition
HEV	Hybrid Electric Vehicle
HIL(S)	Hardware-in-the-Loop (Simulation)
ICE	Internal Combustion Engine
I/O	Input/Output
IPC	Inter-process communication
NEDC	New European Drive Cycle

References

[1] Verein Deutscher Ingenieure, VDI Richtline 2206, Entwicklungsmethodik für mechatronische Systeme – Design methology for mechatronic systems, June 2004.

[2] Dymola, Dynamic Modeling Laboratory, User's Manual, http://www.Dynasim.com: Dynasim AB, 2004.

[3] P. Fritzson, Principles of Object-Oriented Modelling and Simulation with Modelica 2.1. Piscataway, NJ: IEEE Press, 2004.

[4] Microsoft MSDN Library, – Interprocess Communication – Named Pipes, http://msdn.microsoft.com/library/en-us/ipc/base/named_pipes.asp, 2006.

[5] Elmenreich W., Systemnahes Programmieren – C Programmierung unter Unix und Linux, 2002.

[6] National Instruments Document, DAQ - Traditional NI-DAQ User Manual, Version 7.0, April 2003.

[7] D. Simic, H. Giuliani, C. Kral and F. Pirker, Simulation of Conventional and Hybrid Vehicle including Auxiliaries with Respect to Fuel Consumption and Exhaust Emissions, SAE World Congress, Detroit, 2006.

[8] H. Giuliani, C. Kral, J.V. Gragger, F. Pirker, Modelica Simulation of Electric Drives for Vehicular Applications - The Smart Drives Library, ASIM, Erlangen, 2005.