Int J Interact Des Manuf (2014) 8:139–150
DOI 10.1007/s12008-013-0182-3

SHORT ORIGINAL PAPER

A modular architecture for a driving simulator based on the FDMU approach

F. De Filippo · A. Stork · H. Schmedt · F. Bruno

Received: 14 March 2012 / Accepted: 21 February 2013 / Published online: 9 March 2013

Abstract The present paper describes the development of a modular and easily configurable simulation platform for ground vehicles. This platform should be usable for the implementation of driving simulators employed both in training purposes and in vehicle components testing. In particular, the paper presents a first architectural model for the implementation of a simulation platform based on the Functional Digital Mock-Up approach. This platform will allow engineers to implement different kinds of simulators that integrate both physical and virtual components, thus achieving the possibility to quickly reconfigure the architecture depending on the hardware and software used and on specific test case needs. The platform has been tested by developing a case study that integrates a motion platform, some I/O devices and a simple dynamic ground vehicle model implemented in *OpenModelica*.

Keywords Functional Digital Mock-up · Driving simulator · Real time simulation · Virtual reality · OpenModelica

F. De Filippo · F. Bruno (✉)
DIMEG-Dipartimento di Ingegneria Meccanica Energetica e Gestionale, Università della Calabria, Rende, CS, Italy
e-mail: f.bruno@unical.it

F. De Filippo
e-mail: francesco.defilippo@unical.it

A. Stork · H. Schmedt
Fraunhofer-Institut für Graphische Datenverarbeitung IGD, Darmstadt, Germany
e-mail: andre.stork@igd.fraunhofer.de

H. Schmedt
e-mail: hendrik.schmedt@igd.fraunhofer.de

1 Introduction

Driving simulators are used since several years both for videogames and training applications. The most advanced driving simulators include a special hardware, usually called motion platform, which tries to simulate accelerations and vibrations of the car in order to give the user the sensation of being inside a real car [1].

Driving simulators are also used, in the automotive industries, to involve final users in the evaluation of a virtual prototype of a car during the development process. This kind of simulators can be used to test the behaviour of the whole vehicle or to verify how the vehicle dynamics is influenced by a new component (virtual or real) since the early phases of a design process [2,3]. These kinds of simulators have been also developed taking advantage of the *Hardware In the Loop* (HIL) approach. This approach allows to include some of the real devices that influence the vehicle behaviour in the simulation. HIL simulations are increasingly likely to be used in the development and test of complex real-time embedded systems, as they are able to provide an effective platform for developing and testing real-time embedded systems.

Obviously the architecture and the technologies used for the development of a simulator devoted to gaming applications are completely different from the ones used in industry. The first ones use, in general, approximate behavioural models based on few and simple physical laws. Moreover they lack of flexibility because they tightly integrate the mathematical model with the hardware components, making difficult to change the configuration.

The second ones adopt accurate models for each single component considered in the simulation, trying to obtain a global behaviour model as realistic as possible.

The work presented in this paper aims to study the feasibility of a simulation platform for ground vehicles based on an innovative architecture that is able to be quickly reconfigured. The platform should be usable for the implementation of several kinds of driving simulators employed for various tasks like driver training, studies about road safety, design and testing of new components or systems in the automotive field.

The simulation platform should be easily configurable depending on:

(a) the purpose of the specific test-case (e.g.: component development or testing, drivers training, etc.);
(b) the hardware used: motion platform, driving cockpit, devices for visualisation and interaction;
(c) the number and the type of software modules used to simulate the different components of the vehicle;
(d) the possibility to include in the simulation some of the control devices used in the real vehicle, according to the HIL approach.

In other terms, the goal is to create a flexible simulation platform based on a modular architecture that allows to add or replace components, both virtual and real ones depending on the test cases.

In order to fulfil all these requirements, a tool capable of simulating the global functionality of complex systems that can be related to different physical domains (mechanics, electric, electronic, software, etc.) is essential.

This is a well-known problem in the modern industry that continuously require to reduce costs and time-to-market and to increase quality, also for complex and multi-domain objects, e.g. mechatronic components and products.

It stands to reason that designers and engineers need software architectures capable of managing different simulator software in order to furnish a complete virtual prototype that allows to detect and solve problems since the early phase of the design process.

Hence, the integration between solvers for different physical and engineering domains, Virtual Reality applications and I/O devices has been performed in different ways and with different aims.

For example, Tideman et al. [4] and Kanai and Verlinden [5] discuss the importance of prototyping in evaluate stakeholders tastes; in [6] and [7] virtual industrial machines are designed in order to allow workers to train and manufacture process testing; Bruno et al. [8] and Kanai et al. [9] show particular software architectures employed to perform the simulation of virtual products in virtual environments also using real devices.

The simulation platform has been tested by implementing a case study that includes a driving cockpit installed on a Stewart platform [10,11] that, in the final version, will generate the vibrations and the accelerations that give the driver the feeling of being in a real car. The motion platform is located at *Fraunhofer Institut für Graphische Datenverarbeitung* (IGD) in Darmstadt. The simulation platform is based on the *Functional Digital Mock-Up* (FDMU) concept, which represents the evolution of *Digital Mock-Up* (DMU) systems and of classical virtual prototyping [12].

The Fraunhofer Institute developed the FDMU architecture to provide a platform for the integration of a variety of simulators capable of mapping the desired functioning of a product also during the very early steps of its production stages.

2 State of the art

First simulators appeared in the mid-1960s. Earliest notable examples are devices developed by UCLA, GM Styling Staff, Cornell Aero Labs, and Volkswagen (e.g., [13–15]), although they were not able to support vehicle R&D.

Simulators improved during the 1970s and early 1980s, as they were well suited for studying human behaviour, but not yet for vehicle design and development. Some representative examples are simulators developed at the Road and Traffic Research Institute (VTI) in Sweden [16,17], at VPI [18], at the Institute for Perception (IZF-TNO) in the Netherlands, at FAT in Germany, and the HYSIM of FHWA [19].

In the 1980s and 1990s mid-to-high level devices have been installed (e.g. at Daimler Benz in Berlin [20–22] and further improvements have been made to the VTI simulator in Sweden [23] and to the DRI driving simulator [24].

This generation of driving simulators were advanced enough to support both vehicle R&D and studies about driver behaviour.

In the second half of 1990s the National Advanced Driving Simulator (NADS) has been sponsored by the US Department of Transportation's National Highway Traffic Safety Administration (NHTSA) [25]. This simulator is intended to be "world class", providing, among a variety of applications, also vehicle R&D capability.

In the meanwhile, other car companies and other organizations established driving simulators also to support vehicle design and development, but only recently these devices overcome some technical problems (e.g. lack of motion capability, feel characteristics and image accuracy, sickness of simulator, etc.) and became very elaborate and sophisticated driving simulators. Among them, the upgraded simulators at Daimler Benz [26,27], Vertex at Ford [28], BMW [29], Toyota [30], TNO [31], and the current version of the DRI [32] and NADS [33] simulator are notable examples.

Some of the mentioned simulators use very advanced systems of motion in order to reproduce the driving experience as real as possible. They're often employed in researches about traffic safety and human behaviour for studying the influence of factors that can affect attention while driving (e.g. fatigue, illness, optical disturbs, etc.) and to improve cars' safety systems. They're also used to test new car models or new car parts in a safe, economic and perfectly repetitive way.

Focusing on the software control system of simulators, the motion of the hardware device can be managed in different ways.

The simulation can be centred around global vehicle dynamics, using equations and parameters (often empirical parameters) [34] or it can employ a hybrid Motion-Generation method that combines the classic dynamics equations with data acquired from an actual vehicle, providing a very efficient and realistic simulation [35]. These kinds of simulators require a preliminary step for data acquisition or parameters evaluation that makes the simulator able to reproduce a unique vehicle in a unique configuration and so they're quite rigid and not useful for industrial testing applications.

The most advanced driving simulators are controlled by software that performs multi-body simulation in order to accurately evaluate the vehicle dynamics. These software have a modular structure that allows to interface the simulator with real devices and/or other simulators, and to completely reconfigure the specific vehicle, but they are generally written for vehicle simulation (e.g. CAN interface).

Software frameworks like SARTURIS [36], developed by Dresden University of Technology, allows the interactive simulation of technical systems in a virtual reality environment and the cooperative simulation of different models modelled using *Modelica*, the multi-physics modelling language. Each model can be automatically converted in a SARTURIS model thanks to a developed tool. SARTURIS also provides some modules and interfaces for data exchange with lots of devices (CAN port), for 3D visualisation (OpenSceneGraph) and for the motion platform control.

SARTURIS allows to easily add or substitute a *Modelica* module, but it can communicate with others simulators only through the standardized *Functional Mock-up Interface* (FMI) [37,38] as shown in [39].

Others interesting software architecture employed in driving simulators field are represented by control systems used in NADS-1 and DESDEMONA simulator.

The first one uses NADSdyna, a dedicated highly configurable software providing real-time multi-body vehicle dynamics capable of simulating any vehicle [33].

The latter has a totally innovative motion hardware consisting of a fully gimballed system that, as a whole, can move vertically and horizontally and the car mock-up communicates via shared memory to a Matlab-Simulink environment in which the vehicle model is running. The vehicle model is generated in CarSim. The CarSim math models cover the complete vehicle system and its inputs from the driver, ground, and aerodynamics. The models are extensible using built-in VehicleSim commands, MATLAB/Simulink from the Mathworks, LabVIEW from National Instruments, and custom programs written in Visual Basic, C, MATLAB, and other languages.

The state of the art puts in evidence that a noticeable research effort has been made to improve the flexibility and the modular approach of the driving simulators, but most of the proposed solution are based on a specific simulation environment thus reducing the possibility to adapt the simulation architecture to the specific engineering needs. For this reason we have decided to test the use of the FDMU approach to develop a modular platform for driving simulators. We have adopted the framework developed at Fraunhofer Institute [40,41]; the employment of a generic simulation framework can allow to easily use the most appropriate simulator for each component, and to choose the preferred software for each specific aim (e.g. simulation, visualisation, audio reproduction, etc.). Each module will be interfaced, through a custom wrapper, with the central data manager (*Master-Simulator*); this allows to make the most from the capability of the dedicated software components. The FDMU framework also provides a visualisation component that allows the interactive 3D representation of the simulation results. FMDU is based on a Service Oriented Architecture (SOA). A SOA defines some principles and methodologies to design and develop software mechanisms able to manage large problems, in order to find the related solutions by breaking them down into smaller problems. This goal is also pursued by other standardized distributed simulation frameworks like *Distributed Interactive Simulation* (DIS) [42] and its successor, the *High-Level Architecture* (HLA) [43]. Like SOA, these are specifications and not implementations. In particular, HLA overtook some DIS limitations specifying a standard architecture for distributed modelling and simulation; it operates beyond a subnet using less bandwidth and can reduce the development time by providing a flexible data model able to share large quantities of data among applications in a very efficient manner. In order to do this, HLA provides time management and synchronization tools, but it imposes some rules that require some knowledge of components' inner operation.

SOA, instead, benefits from loose coupling, component reuse and scalability, and allows integration of heterogeneous resources and web-wide accessibility across firewall boundaries. As any other SOA, FDMU is suitable to perform the data exchange and the service sharing among different systems and organization, meeting the requirements of secureness and respect of ownership. Furthermore, the FDMU framework is more oriented towards Virtual Prototyping and it has been designed to satisfy the requirements of 3D visualisation and rapid reconfiguration. As better explained in the following section, FDMU provides some additional services, such as algorithms for simulation and time management and tools for 2D/3D visualisation and data analysis to enhance SOA characteristics of interoperability and easy debugging.

With our work we intend to show that a FDMU extension towards the simulation field allows to exploit the high flexibility and the powerful visualisation tool offered by the FDMU architecture in CAE simulation environments, in order to enhance product quality and reducing production times.

3 The FDMU approach

In this section we briefly describe the FDMU framework; further details are available in some of the papers listed in references [12,41,44].

The FDMU tries to give to designers and engineers a tool capable of simulating the global functionality of complex systems that can be related to different physical domains (mechanics, electric, electronic, software, etc.).

In particular, the FDMU framework developed by Fraunhofer Institute has been used. As widely explained in [40] and [41], the framework has a central component, called MasterSimulator, that links different Functional Building Blocks (FBB) containing native behavioural model implemented through either real or virtual components (see Fig. 1).

Each module works independently and communicates just with the *MasterSimulator* through a wrapper application that standardizes the FBB interface. The dedicated wrapper is the only part of the architecture that needs to be implemented (or modified) when a new simulator or a physical component is added (or replaced).

The framework is loose coupled and implemented as a Service Oriented Architecture whose components work as services of a server-client infrastructure. The Web Service standards have been pursued in implementing FDMU API and in HTTP-based SOAP (Simple Object Access Protocol) messages used for signal data exchange, and also to ensure communication secureness, encryption and standardization through firewalls, so that, different companies can collaborate in development without risks for their own confidential information.

The advantage of this approach is a high flexibility for hosts, hardware platform, operating systems and IP domains, allowing the coupling of different simulators also via Internet, even if communication overhead occurs as drawback [45].

In addition the distributed system paradigm of FDMU architecture makes it suitable for large-scale co-simulation scenarios, supporting concurrency and multi-threaded implementations in which thread-safe queues allow for distributed and deadlock-free communication methods. The *MasterSimulator* handles the information in order to get output data from modules and to provide them the necessary input information. Several protocols are available to support complex concept of data transfer: the simplest algorithm handles independently each connection, so preventing dead-locks and maximizing their throughput, but it is also possible to upsample or downsample data, to preserve delays of event succession or to reorder it according to a predefined protocol. As illustrated in Fig. 2, the *MasterSimulator* allocates an output slot for each variable coming from the wrappers and an input slot for any incoming variable required by the behavioural model. The main part of the synchronization is carried out by the *MasterSimulator*, which tries to decouple the co-simulation by running parallel threads and buffering data values in each slot, in order to avoid unnecessary blockings of simulators or waiting for free processing time [12].

The FDMU framework does not require language binding because the coupling passes through each wrapper that follows the WSDL (Web Service Definition Language) specification of the *MasterSimulator* to send information received from its own FBB.

The universal description language SysML [46] adopted for wrapper design allows for a standardized and tool-independent description of interfaces of the native behavior models [41]; incoming and outcoming variables have to be mapped to standardized types in SysML in order to be exchanged with *MasterSimulator*. Thanks to this approach, it is possible to test the consistency of the model at an abstract level, in order to specify value ranges and to check them at run-time.

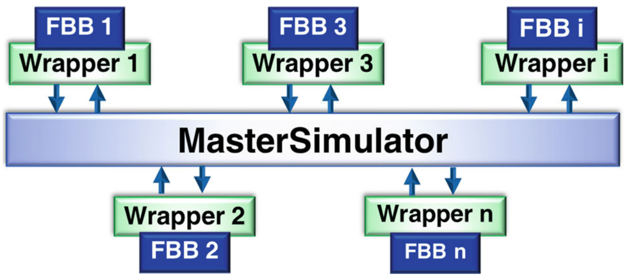


Fig. 1 Basic scheme of FDMU architecture

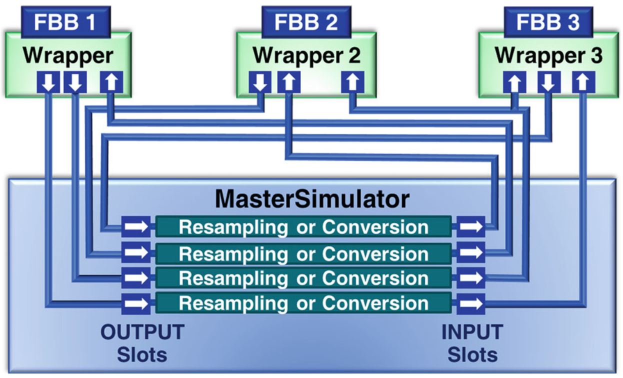


Fig. 2 Data exchange via *MasterSimulator*; slots are named as In/Output according to the wrapper's point of view

Each instance of the wrappers has its unique own text file that correctly configures the connections with the *MasterSimulator* and the FBB. This configuration file contains information about the specific hardware or mathematical model and about the variables and slots which have to be handled. This mechanism increases flexibility and makes this framework particularly suitable for testing different configurations, thanks to the reusability of wrappers and models: it allows the developer to use the same class of wrappers with different instances of FBB that typically require a specific configuration in terms of input and output slots.

The use of FDMU architecture in a driving simulator gives some important advantages with respect to the software control systems mentioned in the state of the art. Like HLA or FMI, FDMU allows to achieve a complex solution by splitting a complex problem in relatively simple partial problems and solving them through different software and/or devices but, as FDMU is expressly oriented towards Virtual Prototyping, it is focused on the concepts of secureness, flexibility, easy reconfiguration and debugging.

In fact, thanks to its service-oriented architecture and to the Web Service Standard employed, it allows each user to test different devices and to choose their favourite software for each analysed domain, easily coupling them via Internet when they are resident on different departments or companies.

No language binding is required and the architecture is capable to accomplish a secure communication between different simulation tools, with no particular request for their open interface, just by a dedicated wrapper.

Furthermore, the FDMU provides a flexible and extendable visualisation component that enhances development and problem solving characteristics of a typical virtual prototyping environment. This interactive visualisation tool allows to control the simulation process and, above all, to display even large DMU models, composed by millions of polygons, giving the possibility to show and observe the movements that are induced by the global system simulation and highlighting, for example, colliding parts or signal source/sink. It is managed independently by the framework, which ensures its consistency with the model and an adequate refresh rate. Finally, thanks to its post-processing functionalities, the user can plot and analyse parameters and variables of the simulation.

4 The architecture of the simulation platform

As a first step, the developed architecture just includes the essential components of a driving simulator. In particular we have a hardware module (made up by a Stewart platform and by a driving cockpit) and a software module represented by a simulator that solves the equations of the mathematical model related to the vehicle dynamics. Thanks to implemented wrappers, the *MasterSimulator* manages the whole simulation and performs the data exchange between modules. The complete architecture also includes a control panel module that is a part of the FDMU visualisation component.

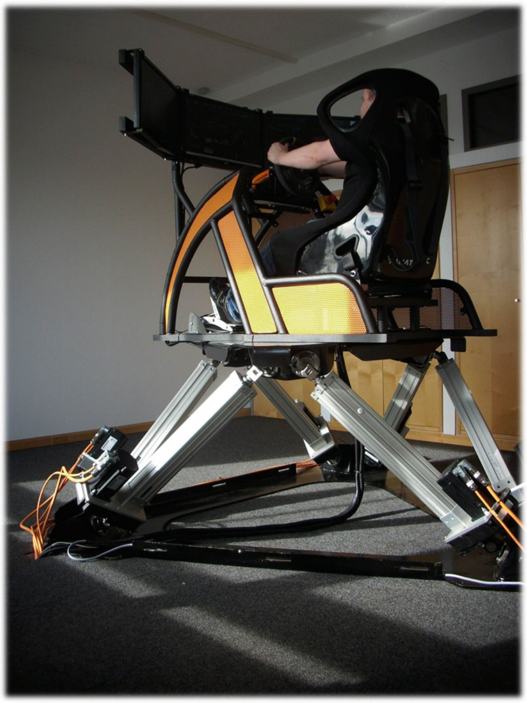


Fig. 3 Motion platform

In this section we treat about hardware and software modules, about the central *MasterSimulator* and, finally, about wrappers needed to integrate each module in the FDMU framework.

4.1 Hardware module

This paragraph describes the hardware devices used in this study and their interfacing capability in sending and/or receiving commands.

As shown in Fig. 3, the hardware module includes a Stewart platform used to move the cockpit and reproduce the accelerations, a cockpit with steering wheel and pedals and a visualisation system made with three LCD displays.

The hardware module provides information about platform state and values of steering wheel, accelerator and brake pedals. At the same time, it receives the joint values calculated by the wrapper to move the platform according to simulation results (see Table 1).

4.1.1 Stewart platform

The Stewart platform is provided by OTLO (see Fig. 4) and it has six ball screw actuators, each one independently controlled. Hence the platform has six degrees of freedom that

Table 1 Variables exchanged by hardware module with *MasterSimulator*

Incoming variables	Actuator extensions (six degrees of freedom evaluated by the wrapper starting from vertical translation, roll angle and pitch angle values given by software module)
Outcoming variables	Steer angle
	Throttle pedal angle
	Brake pedal angle

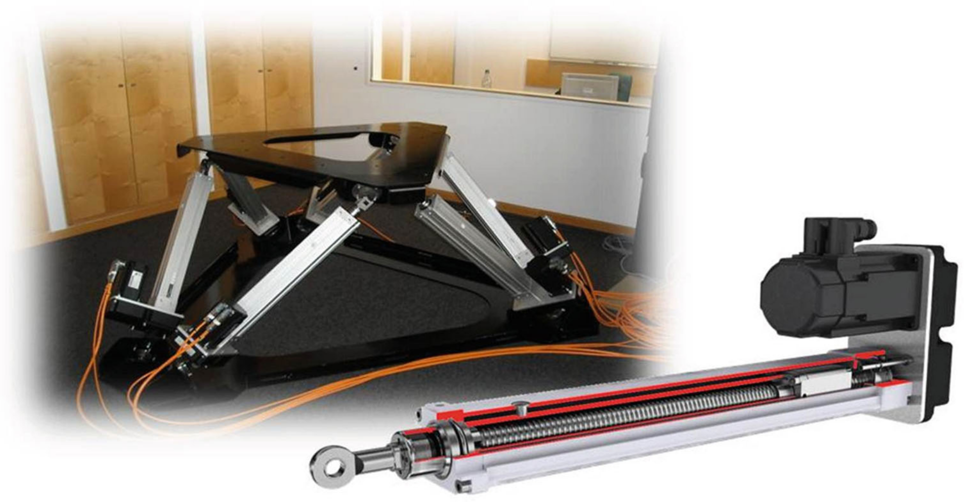


Fig. 4 Stewart platform and ball screw actuator

can be controlled by imposing the extensions of the actuators at a maximum speed of about 2 m/s.

The robot is connected with an industry-standard computer (OTLO VR Systeme), shown in Fig. 5, which controls all its components (actuators, electric supply, encoders, etc.) and contains process data acquisition and control modules.

The control computer can exchange data over a network through UDP socket.

4.1.2 *Steering wheel and pedals*

The cockpit has been equipped with a FANATEC kit that includes a steering wheel and three pedals (see Fig. 6). The kit is configured as a multi-axis joystick and it comes with DirectX drivers. It can be connected to a PC through a USB port and the data can be read using DirectX API.

This kit works on three axes: one for the steering wheel, another for throttle and brake pedals and the last one for clutch pedal. Throttle and brake give, respectively, positive and negative values on the same axis. It has got also several buttons that may be assigned to custom functions.

4.2 Software module

The software module simulates the car dynamic model using input values received from *MasterSimulator* and it sends back results about the dynamic parameters.

Fig. 5 Industry-standard computer OTLO VR Systeme

Fig. 6 Steering wheel and pedals by FANATEC

We have adopted a simplified dynamic model of a car written in *Modelica* language and simulated using *OpenModelica* (OM). OM provides an interactive simulation subsystem, called OM *Interactive* (OMI), which allows a user-interactive and time synchronous simulation.

OMI is part of the simulation runtime core and results in an executable simulation application, which contains the full *Modelica* model as C/C++ code with all required equations, conditions and different solvers to simulate a whole system or a single system component [47].

This executable application communicates with external applications via message passing using TCP/IP.

At this stage of the work we've implemented a simplified three degrees-of-freedom (*dof*) full-car model like the one shown in Fig. 7.

After setting different parameters, the simulator evaluates the vertical translation and pitch and roll angles that result from the displacement given for each wheel (*d-fr*, *d-fl, d-rl* and *d-rr*).

The same three dynamic parameters are also evaluated through some qualitative relations using the data received from cockpit input devices (steer, acceleration and brake pedal angles) and by estimating longitudinal speed and acceleration. In this simplified model, the final motion set that has to be applied to the Stewart platform is just obtained as the superposition of the effects due to road roughness and user input (actually the effects influence each other). This behavioural model is very simple (it is not intended to ensure high accuracy), and it has been implemented just for testing connection and communication capability of the whole system using a standard commercial PC (i7-Q720 @ 1.76 GHz processor, 6 GB RAM); in *Modelica* environment we are able to obtain a 1 millisecond resolution spending about 70 ms of wall-clock time for each second of model simulation time.

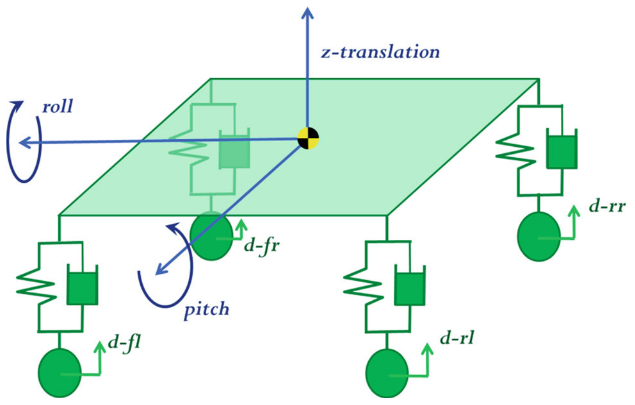


Fig. 7 Three-*dof* full-car model

The whole set of parameters and variables managed by the model is summarized in Table 2; according to *OpenModelica* conventions, we define as "variable" those properties that have to be evaluated or received as input, and as "parameters" the ones that are assumed to be constant during the simulation, but that could be also interactively changed by the user, through the configuration interface of the simulator, in different simulations or in different phases of the same simulation without editing the model.

It should be noted that, in this version, the wheel displacements are assigned by using a random periodic function, but in future they will be extracted from the virtual simulation scene.

We will also implement the possibility to change some simulation parameters at run-time. In the specific case, the simulation parameters represent the most relevant characteristics of a vehicle (e.g.: mass, dimensions, spring and dumping suspensions coefficients, etc.) but they could include information about driving assistance systems, external environment, components wear, etc.

Table 2 Parameter and variable set handled by the simulator

Parameters	Body weight
	Car track and wheelbase
	Moment of inertia about pitch axis
	Moment of inertia about roll axis
	Centre of Gravity position
	Spring constants
	Dumper constants
	Suspension lengths
	Additional forces on suspensions
Simulator internal variables	4 × wheel displacement
	Velocity along z
	Acceleration along z
	Angular velocity about roll axis
	Angular acceleration about roll axis
	Angular velocity about pitch axis
	Angular acceleration about pitch axis
Incoming variables (from *MasterSimulator*)	Steer angle
	Throttle pedal angle
	Brake pedal angle
Outcoming variables (to *MasterSimulator*)	Vertical translation

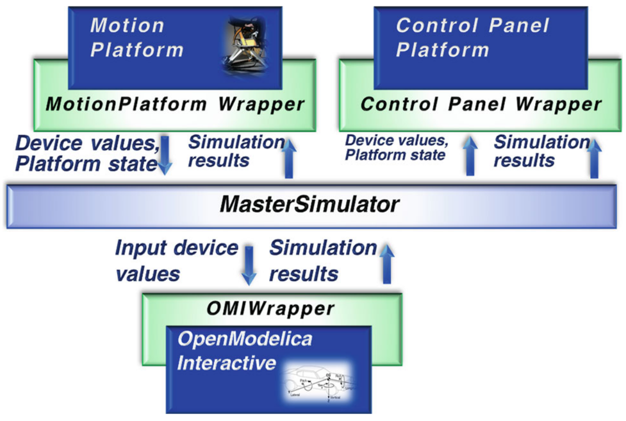


Fig. 8 Scheme of implemented control architecture

4.2.1 FDMU MasterSimulator

The data exchange between software and hardware module is realized thanks to the FDMU *Mastersimulator*, which loads the needed modules and launches the global simulation, moving the platform according to the dynamic parameters evaluated by the mathematical model that processes the user inputs.

The complete scheme is visible in Fig. 8.

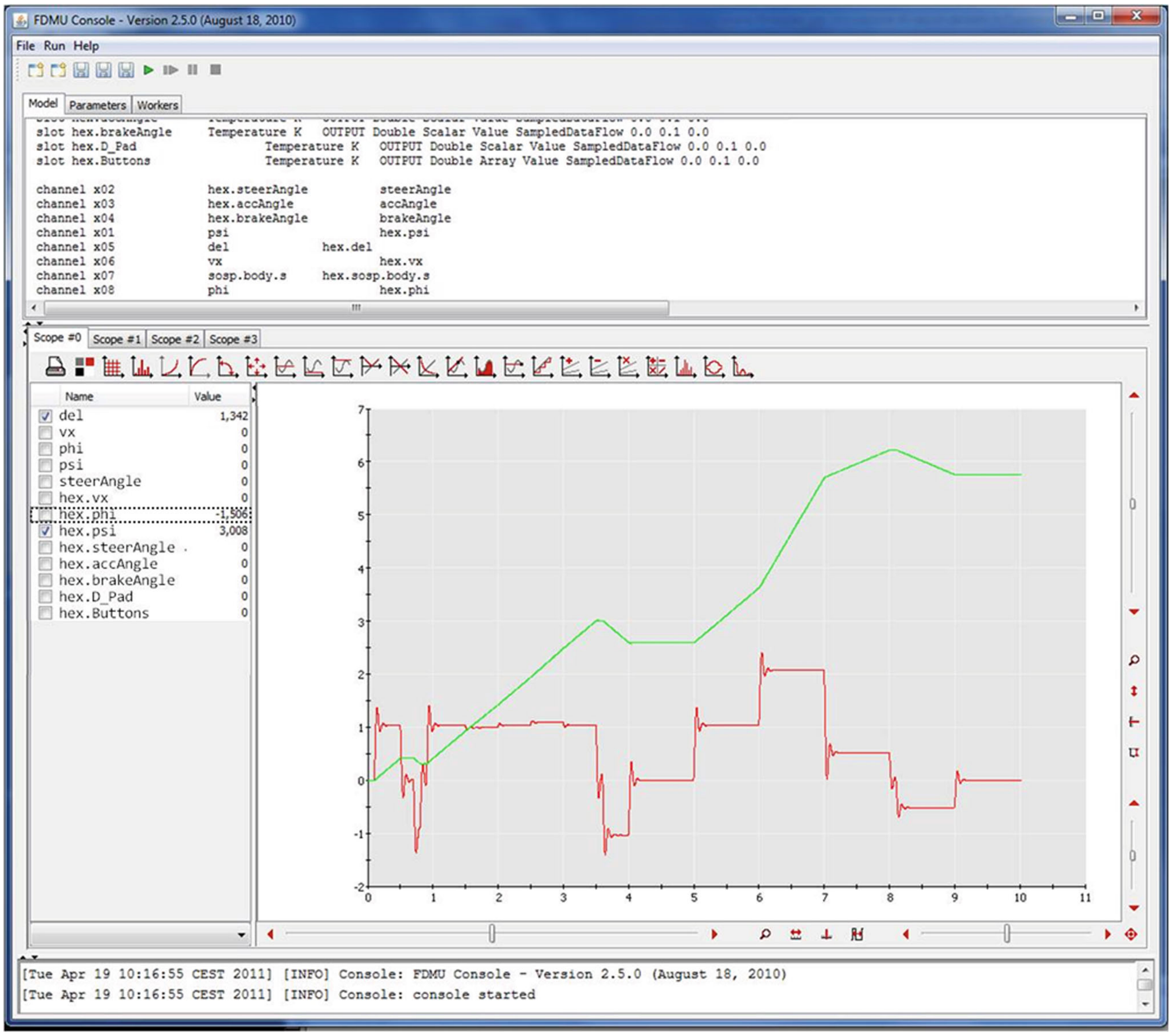
While the simulation is running, it is possible to check every variable or parameter through the Control Panel module linked to the *MasterSimulator*. This module provides a graphical interface (illustrated in Fig. 9) that allows the user to launch and/or stop the simulation and to monitor each simulation variable. In the future, through the control panel and the visualisation tool, it will be possible to set some of the simulation parameters, e.g.: data about the scenario, characteristics of the dynamic model, etc., and to interactively visualise the simulated 3D model.

4.3 Implementation of the wrappers

Both the hardware and the software module need to exchange data with the *MasterSimulator*; in order to perform data exchange two wrappers have been implemented. Implemented wrappers are executable applications written in C++ language and launched by the central framework. They handle the related module and create the needed slots.

In particular, the motion platform wrapper links the hardware modules with *MasterSimulator*. It acquires input device values and information about platform state and writes them on the specific slots of the*MasterSimulator*. The *MasterSimulator* makes the simulation results available for the motion platform wrapper that reads and elaborates them in order to calculate joint variables of actuators. The wrapper, in fact, performs the resolution of the inverse kinematics, determining the translation that each actuator has to perform in order to achieve the pose of the platform evaluated by the solver. Figure 10 shows the data flow between the motion platform and the *MasterSimulator.*

The motion platform wrapper allows to exchange data with the robot and to control its state and its movements. The robot is controlled by a dedicated calculator that exchanges



```
slot hex.brakeAngle     Temperature K   OUTPUT Double Scalar Value SampledDataFlow 0.0 0.1 0.0
slot hex.D_Pad                  Temperature K   OUTPUT Double Scalar Value SampledDataFlow 0.0 0.1 0.0
slot hex.Buttons                Temperature K   OUTPUT Double Array Value SampledDataFlow 0.0 0.1 0.0

channel x02             hex.steerAngle          steerAngle
channel x03             hex.accAngle            accAngle
channel x04             hex.brakeAngle          brakeAngle
channel x01             psi                     hex.psi
channel x05             del             hex.del
channel x06             vx                      hex.vx
channel x07             sosp.body.s     hex.sosp.body.s
channel x08             phi                     hex.phi
```

	Name	Value
✓	del	1,342
	vx	0
	phi	0
	psi	0
	steerAngle	0
	hex.vx	0
	hex.phi	-1,506
✓	hex.psi	3,008
	hex.steerAngle	0
	hex.accAngle	0
	hex.brakeAngle	0
	hex.D_Pad	0
	hex.Buttons	0

```
[Tue Apr 19 10:16:55 CEST 2011] [INFO] Console: FDMU Console - Version 2.5.0 (August 18, 2010)
[Tue Apr 19 10:16:55 CEST 2011] [INFO] Console: console started
```

Fig. 9 Graphic interface of the control panel

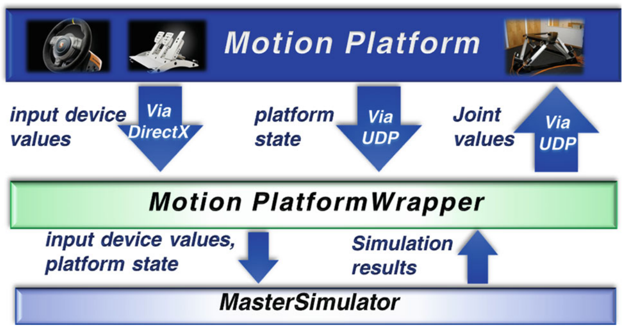


Fig. 10 Scheme of data exchanged by motion platform wrapper

data with external applications (in this case, with the motion platform wrapper) through sockets, using an UDP/IP communication protocol.

The motion platform wrapper uses *DirectX* libraries to acquire data by the input devices (steering wheel and pedals), thus it can be reused also with other *DirectX* compatible input devices.

The OMI wrapper reads input device values from *MasterSimulator* and sends these data to the OMI simulator, which evaluates the pose of the motion platform (limited to pitch and roll angle and vertical position) and sends it to the wrapper.

Finally the OMI wrapper writes the simulation results on the slots of *MasterSimulator*, which make them available for the other modules.

Figure 11 illustrates data exchange between OMI and *MasterSimulator.*

The OMI wrapper launches the *Modelica* model that simulates the car dynamics and controls the simulation process.

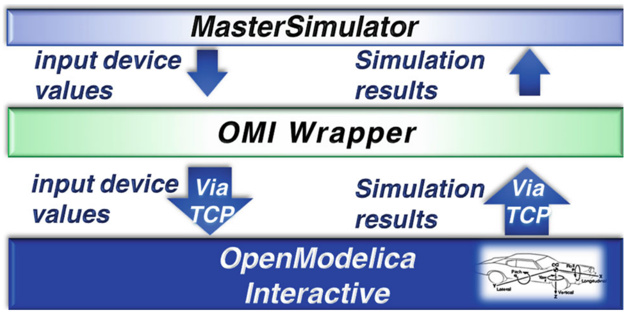


Fig. 11 Scheme of data exchanged by OMI wrapper

The OMI wrapper and the related module exchange information through different sockets using TCP/IP protocol.

Both the motion platform and the OMI wrappers initialize all the slots required by *MasterSimulator* using their own configuration file.

It is important to point out that wrappers are totally reusable and they make the platform usable for simulating any kind of vehicle (not only terrestrial) with different levels of detail, just by coupling a different *Modelica* model with OMI Wrapper and editing the configuration file; the same simple operation is required to be done on the platform wrapper configuration file, if different input devices are needed.

5 Conclusions

With the developed architecture, we verified the feasibility of a control architecture for a driving simulator based on the FDMU approach and technologies. In order to achieve this goal, we have elaborated a simplified dynamic ground vehicle model and we have implemented wrappers to interface the central framework both with hardware and software modules.

For the first time the FDMU framework has been integrated with a physical component (i.e. the motion platform), giving us a positive feedback about the effectiveness of the proposed approach.

The implemented modules have been used to conduct the first tests using a standard PC (i7-Q720 @ 1.76 GHz processor, 6 GB RAM). The real-time simulations have been performed with *Modelica* simulator, the two *Wrappers* and the *MasterSimulator* running on the same machine, reaching a complete exchange time of about 50 ms, allowing to refresh the platform position 20 times per second; the fluidity of platform movement is anyway ensured by its own refresh rate. No time management has been harnessed at this step. Future works will be intended to enhance communication performances, now limited by TCP/IP sockets required on the simulator side, and provide further information about time frequency and average time delay realized in order to confirm good results already obtained in other applications [41,44]. Other tests will be performed to verify the different available time management methods and to better check reliability of HIL integration according to bandwidth characteristics.

The implemented architecture has got some interesting characteristics related, above all, to the development process of vehicles and/or vehicles components:

- Possible integration of both virtual and real components;
- Easy and quick integration/substitution of simulation components;
- Rapid integration of new classes of components;
- Real-time monitoring of simulation parameters and events visualisation.

The FDMU approach gives the possibility to quickly change the number and the configuration of the FBBs connected to the MasterSimulator. So, for example, the same OMI wrapper could be used also to include in the architecture other FBBs containing different *Modelica* models (e.g.: thermal, electric, electronic, etc.) thus obtaining a more detailed and reliable virtual prototype.

It represents a powerful way to create a flexible simulation environment using the preferred hardware for the physical components and the most suitable software for each physical domain, ensuring the complete control of the simulation and of its parameters.

Furthermore, the software architecture implemented for the platform allows to simulate different kinds of vehicles, just by replacing the *Modelica* model and/or the input devices and editing wrappers and *MasterSimulator* configuration files, thus achieving the possibility to quickly configure the architecture according to specific test case needs.

Its capabilities of early inspection and problem pinpointing provide a powerful tool for testing each hardware component or software configuration at any development stage, always taking into consideration the positive contribution on the whole product functionality. These characteristics are essential in every industry field, and in particular in the automotive engineering area, because they allow to improve quality, reliability and appeal on customers too, reducing the time-to-market.

In the future, the architecture will be extended with new components to enrich the visual and audio simulation and the detail level of the mathematical model, in order to test it in different and more challenging scenarios.

References

1. Bergamasco, M., Avizzano, C.A., Angerilli, M., Carrozzino, M., Facenza, G., Frisoli, A.: Fork-lift truck simulator for training in industrial environment. In: Proceedings of Virtual Concept 2005 (2005)

2. Freeman, J.S., Watson, G., Papelis, Y.E., Lin, T.C.: The Iowa Driving Simulator: An Implementation and Application Overview (1996)
3. Reymond, G., Heidet, A., Canry, M., Kemeny, A.: Validation of Renault's dynamic simulator for Adaptive Cruise Control experiments. In: Proceedings of the Driving Simulation Conference DSC'2000. Paris, France, pp. 181–192 (2000)
4. Tideman, M., van der Voort, M.C., van Houten, F.J.A.M.: A new product design method based on virtual reality, gaming and scenarios. Int. J. Interact. Design Manuf. **2**, 195–205 (2008)
5. Kanai, S., Verlinden, J.: Advanced prototyping for human-centered design for information appliances. Int. J. Interact. Design Manuf. **3**, 131–134 (2009)
6. Acal, A.P., Lobera, A.S.: Virtual reality simulation applied to a numerical control milling machine. Int. J. Interact. Design Manuf. **1**, 143–154 (2007)
7. Sghaier, A., Soriano, T.: Using high-level models for modeling industrial machines in a virtual environment. Int. J. Interact. Design Manuf. **2**, 99–106 (2008)
8. Bruno, F., Caruso, F., Li, K., Milite, A., Muzzupappa, M.: Dynamic simulation of virtual prototypes in immersive environment. Int. J. Adv. Manuf. Technol. **43**(5–6), 620–630 (2009)
9. Kanai, S., Miyashita, T., Tada, T.: A multi-disciplinary distributed simulation environment for mechatronic system design enabling hardware-in-the-loop simulation based on HLA. Int. J. Interact. Design Manuf. **1**, 175–179 (2007)
10. Gough, V.E.: Contribution to discussion of papers on research in automobile stability, control and tyre performance. In: Proceedings of Auto Div. Inst. Mech. Eng., pp. 392–394 (1956–1957)
11. Stewart, D.: A Platform with Six Degrees of Freedom. In: Proceedings of Institution of Mechanical Engineers (UK), Vol. 180 (Pt 1, No 15) (1965–1966)
12. Schneider, P., Clauß, C., Enge-Rosenblatt, O., Schneider, A., Bruder, T., Schäfer, C., Voigt, L., Stork, A., Farkas, T.: Functional digital mock-up—more insight to complex multi-physical systems, Bonn (2010)
13. Wojcik, C.K., Hulber, S.F.: The driving simulator—a research tool. ASME Paper 65-WA/HUF-13 (1965)
14. Beinke, R.E., Williams, J.K.: Driving simulator. In: Paper presented at the General Motors Corporation Autootive Safety Seminar (1968)
15. Lincke, W., et al.: Simulation and measurement of driver vehicle handling performance. SAE Paper 730489 (1973)
16. Nordmark, S., et al.: A moving base driving simulator with wide angle vision system. In: 64th Annual Meeting, Transportation Research Board, Washington D.C. (1985)
17. Nordmark, S.: VTI driving simulator—mathematical model of a four-wheeled. Swedish Road and Traffic Institute, VTI No. 267A (1984)
18. Wierwille, W.W.: Driving simulator design for realistic handling. In: Proceedings of the Third International Conference on Vehicle System Dynamics, Swets and Zeitlingerand, Amsterdam (1975)
19. Alicandri, E.: HYSIM: the next best thing to being on the road. Public Roads. Winter **57**(3): 19–23 (1994)
20. Drosdol, J., Panik, F.: The Daimler-Benz driving simulator a tool for vehicle development. SAE Paper 850334 (1985)
21. Hahn, S., Käding, W.: The Daimler-Benz driving simulator—presentation of selected experiments. SAE Paper 880058 (1988)
22. Käding, W.: The advanced Daimler-Benz driving simulator. SAE Paper 950175 (1995)
23. Nordmark, S.: The new Trygg Hansa truckdriving simulator: an advanced tool for research and training. In: Swedish Road and Transport Research Institute, VTI Reprint 187 (1992), 6 pages, Proceedings of the International Symposium on Advanced Vehicle Control 1992 (AVEC '92), Yokohama, Japan (1992)
24. Weir, D.H., Bourne, S.M.: An overview of the DRI driving simulator. SAE Paper 950173 (1995)
25. Stall, D.A., Bourne, S.: The national advanced driving simulator: potential applications to ITS and AHS research. In: Proceedings of the 1996 Annual Meeting of ITS America (1996)
26. Breuer, J.J., Kaeding, W.: Contributions of driving simulators to enhance real world safety. In: Proceedings of the Driving Simulator Conference Asia/Pacific 2006, Tsukuba, Japan (2006)
27. New driving simulator taken into operation in Sindelfingen: Investment in cutting-edge technologies. http://www.daimler.com/dccom/0-5-658451-1-1338848-1-0-0-0-0-1-8-7165-0-0-0-0-0-0-0.html. Accessed 4 Mar 2013
28. Greenberg, J., Artz, B., Cathey, L.: The effect of lateral motion cues during simulated driving. In: Proceeding of the Driving Simulator Conference North America 2003, Dearborn (2003)
29. Huesmann, A., Ehmanns, D., Wisselman, D.: Development of ADAS by means of driving simulation. In: Proceedings of the Driving Simulator Conference Europe 2006, Paris (2006)
30. http://www2.toyota.co.jp/en/news/07/1126_1.html. Accessed 4 Mar 2013
31. Feenstra, P.J., Wentink, M., Roza, Z.C., Bles, W.: Desdemona, an alternative moving base design for driving simulation. In: Proceedings of the Driving Simulator Conference North America: September 2007. Iowa City, Iowa (2007)
32. http://www.dynres.com/prod_drivingsimulators.html. Accessed 4 Mar 2013
33. National Advanced Driving Simulator Overview. http://www.nads-sc.uiowa.edu/. Accessed 4 Mar 2013
34. Ambrož, M., Prebil, I.: i3Drive, a 3D interactive driving simulator. IEEE Comput. Graph. Appl. **30**(2), 86–92 (2010)
35. Cha, M., Yang, J., Han, S.: A Hybrid Driving Simulator with Dynamics-Driven Motion and Data-Driven Motion. SIMULATION (2008)
36. Frenkel, J., Schubert, C., Kunze, G., Jankov, K.: Using Modelica for interactive simulations of technical systems in a virtual reality environment. In: Proceedings 7th Modelica Conference, Como, Italy (2009)
37. http://www.modelisar.org/. Accessed 4 Mar 2013
38. Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Clauß, C., Elmqvist, H., Junghanns, A., Mauss, J., Monteiro, M., Neidhold, T., et al.: The functional Mockup interface for tool independent exchange of simulation models. In: Proceedings of the 8th International Modelica Conference, Dresden, Germany (2011)
39. Gruening, T., Kunze, G., Katterfeld, A.: Simulating the working process of construction machines. In: 3rd International Conference & Exibithion BulkSolids Europe 2010, Glasgow, Scotland (2010)
40. Stork, A., Thole, C.A., Klimenko, S., Nikitin, I., Nikitina, L., Astakhov, Y.: Simulated reality in automotive design. In: International Conference on Cyberworlds, Hannover (2007)
41. Stork, A., Wagner, M., Schneider, P., Bruder, T., Hinnerichs, A.: Functional DMU: co-simulation of mechatronic systems in a virtual environment. In: ASME, editor. Proceedings of the ASME 2011 World Conference on Innovative Virtual Reality, Milan, Italy, pp. 193–198 (2011)
42. IEEE Standard for Distributed Interactive Simulation—Application Protocols. Distributed Interactive Simulation Committee of the IEEE Computer Society. IEEE Std 1278.1a-1998 (Supplement to IEEE Std 1278.1-1995) (1998)
43. DoD High Level Architecture (HLA) for Simulations. Department, U. S. Defense, Under Secretary of Defense for Acquisition and Technology, USD (A&T), memorandum (1996)
44. Schneider, P., Clauß, C., Schneider, A., Stork, A., Bruder, T., Farkas, T.: Towards more insight with functional digital mockup. In: Proceedings of the 4th EASC 2009 European Automotive Simulation Conference, Munich, Germany, pp. 325–336 (2009)

45. Enge-Rosenblatt, O., Clauß, C., Schneider, A., Schneider, P.: Functional digital mock-up and the functional mock-up interface—two complementary approaches for a comprehensive investigation of heterogeneous systems. In: International Modelica Conference, Dresden (2011)
46. SysML. http://www.sysml.org/. Accessed 4 Mar 2013
47. Fritzson, P., Pop, A., et al.: OpenModelica System Documentation (Version for OpenModelica 1.5) (2010)