Using Modelica for Interactive Simulations of Technical Systems in a Virtual Reality Environment

Jens Frenkel[1] Christian Schubert[1] Guenter Kunze[1] Kristian Jankov[2]
Dresden University of Technology, Institute of Mobile Machinery and Processing Machines
Muenchner Platz 3, D-01066 Dresden, Germany
CNH Baumaschinen GmbH
Staakener Strasse 53-63, D-13581 Berlin (Spandau), Germany

Abstract

Simulation has become an essential tool in the development of construction machinery. In addition to the validation of technical features, the assessment of man-machine interaction has become more important within complex working environments. In cases where most attention is paid to the human as the operator, simulations have to fulfil special requirements. Allowing the user to interact with the system implies the need for real time simulation as well as flexible hardware integration and a powerful visualisation. Therefore a modular software framework called SARTURIS[3] has been developed meeting all these requirements. In order to support flexible multi-domain modelling the Modelica language is being used. This paper presents SARTURIS and its applications, focusing on the integration of Modelica based on OpenModelica using the example of a wheel loader. Since OpenModelica is not yet able to deal with the Modelica Multibody library, a Python-based tool called PyMbs has been developed. It allows comfortable description of multibody systems and export to Modelica code as well as other formats.

Keywords: real time simulation; construction machinery; virtual reality; OpenModelica;

1 Introduction

A main focus of research is to study the impact of the operator on mobile machinery. Due to the low level of automation in such machines, the stress on components during usage heavily depends on the way a machine is operated. Studying this influence provides essential information needed during the design process of such a machine. Obtaining these information from experimental data requires a real prototype and increases demands on cost and time. Furthermore, it is extremely difficult to provide equal conditions for each experiment which makes results hardly comparable. Using simulation instead is a more efficient way of achieving those results without facing the aforementioned problems. It also can be used for studying dangerous manoeuvres without endangering man and machine. Thus, simulation proves to be a valuable tool (see Figure 1).

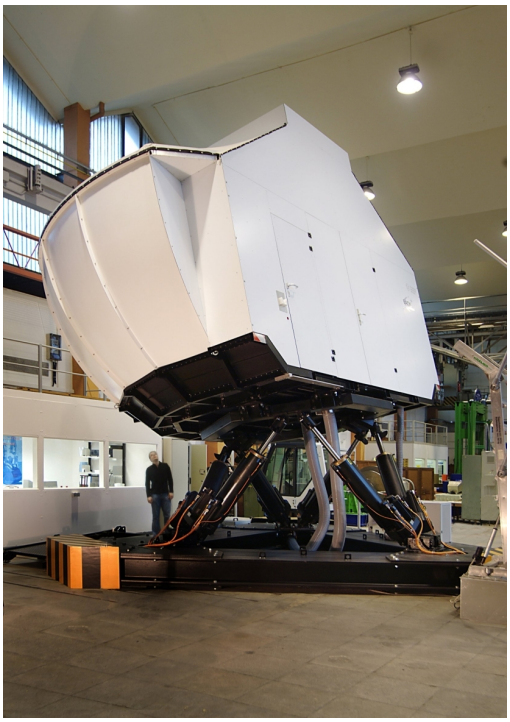

Figure 1: Motion platform

[1]TU Dresden http://tu-dresden.de/bft
[2]CNH Baumaschinen GmbH http://www.cnh.com
[3]SARTURIS BMBF support code: 01ISC24A
. INPROVY BMBF support code: 02PC1110

DOI: 10.3384/ecp09430080

Only if a sophisticated model of the whole machine is implemented, significant results can be obtained. Such models always involve different domains like mechanics, hydraulics and control. The Modelica language is ideally suited for describing these models, since it has been designed to support multi-domain modelling [6]. Moreover, its object-oriented approach allows reuse and substitution of submodels, simplifying the creation of a model [8].

The inclusion of an operator, however, is very challenging. Obtaining a mathematical operator model, which has to be able to react and decide, is virtually impossible. Therefore Virtual Reality (VR) is the only viable option leading to a simulation with a "human in the loop". There are a lot of publications describing the use of VR for analysing the influence of drivers' behaviour, e.g. [1][2][3][4]. Furthermore, the development of the human-machine interface is supported by new methods of VR technologies. The articles point out that the current adoption of these technologies in the industrial sector is rather low [5]. Current simulation systems support the modelling and use of VR technologies only to a minor degree. There is no Modelica tool known to the authors that is specialised in interactive VR simulations.

A tool is needed that not only carries out calculations in real time, but also offers realistic graphics as well as a support for a large variety of input and output devices. The simulation framework SARTURIS, specifically developed at our institute towards interactive VR simulation, meets all the aforementioned requirements including support for Modelica as primary modelling language.

2 SARTURIS

The simulation framework SARTURIS[1] has been developed at Dresden University of Technology in cooperation with industrial partners within a publicly funded (BMBF) research project [10] [11] [12]. SARTURIS allows interactive simulation of technical systems in a virtual reality environment. In order to achieve the best compromise between performance, portability, and development methodology, SARTURIS is based on C++ and uses freely available libraries.

SARTURIS itself is merely a slim application featuring a module loader establishing a framework for individual software components and thereby enabling reusability (Figure 2). Efficient creation of new software components is guaranteed through the use of Model Driven Architecture (MDA). The interaction between these modules along with their parameterisation is specified in XML-files. Each software component has its own XML type definition describing its usage and configuration as well as the interaction with other components. Thus automated syntax checking or even code completion is available. XML files are either written as plain text or assembled using a graphical user interface (GUI).

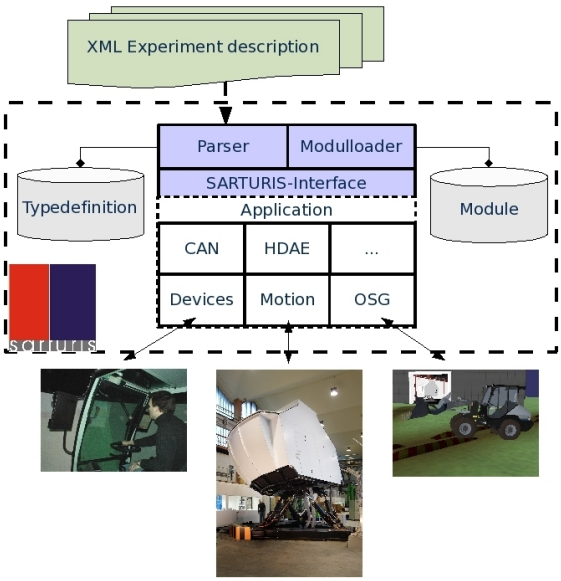

Figure 2: Sarturis Framework

Software components belonging to the same field of functionality are encapsulated within the same module. For instance the module OpenSceneGraph (OSG) [13], see Figure 2, contains all necessary functions to achieve a realistic 3D Visualisation. A comprehensive set of modules has already been created. Graphical User Interfaces can be defined within the XML files by using a module referencing the GIMP-Toolkit (GTK) [14]. In order to integrate miscellaneous input and output devices a module featuring Controller Area Network (CAN) communication has been implemented [15]. Even our motion platform (Figure 1) can be operated via SARTURIS by means of a corresponding module.

To support the development of additional modules several different interfaces have been designed in C++. Every implementation of a technical model inherits from an according interface like *HDAESystem*.

[1]BMBF support code: 01ISC24A

3 Integrating Modelica Models into SARTURIS

3.1 Initial Situation

Every technical system which is to be simulated within SARTURIS forms a module on its own, that contains a class which inherits from the interface *HDAESystem*. Therefore it was necessary to translate the system equations directly into C++ code. Although C++ is extremely powerful as a programming language it is ill-suited for modelling purposes. Hence, the integration of a new model was very tedious and error-prone. Moreover, the resulting code was neither reusable nor maintainable.

The Modelica language on the other hand facilitates convenient modelling of technical systems. The model description is reusable and easily maintainable through its equation-based and object-oriented approach. Furthermore it is very flexible due to the acausal description.

In order to combine the strengths of Modelica and the capabilities of SARTURIS, a transformation of Modelica code into C++ code was needed. OpenModelica [22] proves to be the best solution, since it is able to translate Modelica models into C code. Beyond that, the usage of open-source software is very beneficial to universities since it offers great flexibility and can be used for teaching. Furthermore, every user has the opportunity to get involved in the development of the simulation software through the OpenModelica Consortium.

3.2 OpenModelica Code Export

Before discussing the integration of the OpenModelica Code Export into SARTURIS, a brief introduction on how the OpenModelica Compiler (OMC) translates Modelica code into a simulation shall be given. This process is divided into different stages which are shown in Figure 3. First, the given coherent Modelica model is translated into a flat model where all of the object-oriented structures are removed yielding a system of differential and algebraic equations. Next, this system of equations is analysed and optimised with regards to numerical integration. Consequently the resulting system of equations is passed to a code generator which converts the optimised system of equations into C Code.

The so-called C Code Export yields the following files:

- Text file (*init file*) containing all initial values and information about the system and the solver.
- C source code file (*c model*) containing all equations arranged so that they can be readily used with the supplied solver.
- C source code file (*c model functions*) containing all functions both external and internal which are used in the model.
- precompiled libraries (*sim libs*) containing all model independent functions needed for linking

The *c model* and the *c model functions* are compiled and linked against the *sim libs* using an appropriate C/C++ Compiler resulting in a stand alone programme which runs the simulation and stores the result in a text file. This file contains all the values of the states, their derivations and the algebraic variables that occured during the simulation.

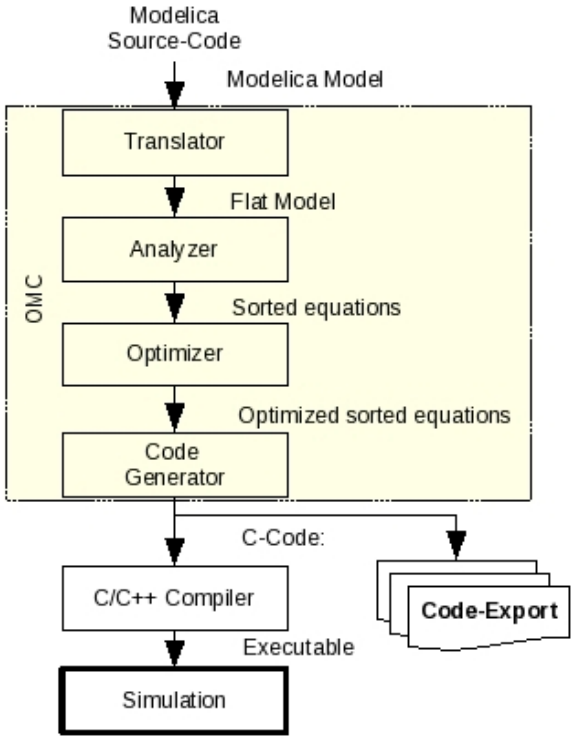

Figure 3: Translation Stages from Modelica Code to a Simulation. According to [9], p. 10

3.3 Integrating the OMC Code Export into SARTURIS

To transfer C code generated by the OpenModelica Compiler automatically into C++ code which can readily be used within SARTURIS as a module, the following two steps have to be carried out.

In the first step information about the model has to be gathered in order to generate the C++ class interface along with its XML type definition as described in section 2. Following information is needed.

1. names of all
   (a) inputs
   (b) outputs
   (c) parameters
   (d) states
   (e) algebraic variables
2. default values of all parameters
3. initial values of all states

The *init file*, as part of the OMC Code Export, provides most of the information except 1.a and 1.b. All information but 2. and 3. is stored within the *c model*. Since there is no single file containing all the information needed, both, the *init file* and the *c model* have to be evaluated.

The second step is to convert the C code generated by the OMC into C++ code implementing a SARTURIS module. A major requirement is to minimise changes within the C code. Ideally, it should be possible to use it without any modifications at all by encapsulating it into a wrapper class. Unfortunately this does not seem to be possible as discussed in the following sections. Furthermore, the system of differential equations and the solver should be separated. Thus results of different solvers can be compared without having to recompile the model. So far the DASSL-solver which comes with OpenModelica and some standard solvers like an Explicit Euler and a fourth order Runge Kutta solver have been implemented. A current student project is dealing with the integration of the SUNDIALS package [16].

3.4 Automation

A tool called OpenModelicaToSarturis (OM2S), see (Figure 4), has been developed which automates the procedure outlined in the previous subsection. It enables the user to generate a SARTURIS module without writing a single line of C++ code. Thus, models developed in Modelica can be easily used within SARTURIS for interactive VR simulations.

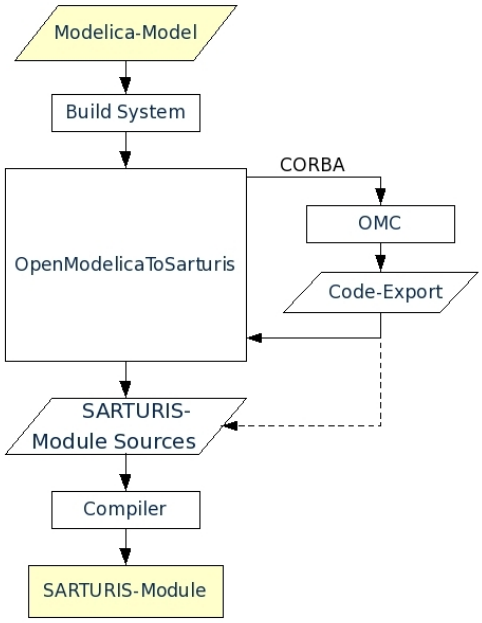

Figure 4: OpenModelicaToSarturis: Automated Translation of Modelica Models into a SARTURIS Module

The whole compilation process from a Modelica model to a SARTURIS module is coordinated by CMake [17]. Thereby, custom build rules can be defined easily and it is possible to detect utility programmes, libraries and include directories in a platform neutral manner. It generates makefiles and workspaces which can be used with any supported compiler. At the beginning of the compilation process OM2S is started which then launches the OMC. Communication between OM2S and OMC is achieved via CORBA, offering a convenient interface to trigger the translation of a Modelica model. Subsequently, OM2S turns the C code into C++ code implementing a SARTURIS module. Simultaneously, a sample SARTURIS configuration is generated featuring a diagram for each state as well as sliders for each input value. After the compilation process is finished, SARTURIS can be launched with the sample configuration [18]. It allows validating the results immediately and it may also be used as a template for more complex settings.

3.5 Difficulties And Suggested Solutions

This section describes the difficulties encountered during the integration of the OMC Code Export into SARTURIS and the measures taken to overcome them. In addition, suggestions to improve the OMC Code Export regarding usability are given.

3.5.1 Gathering System Information

Gathering system information needed for the class interface as well as the XML type definitions, requires a parsing of the *init file* and the *c model*. Parsing the *init file* can be achieved using internal functions of the *sim libs* which are called at every start of a simulation run. Analysing the C code however proves to be much more challenging. The current implementation reads the C code line by line looking for unique keywords. All names of the state variables, for instance, are stored within the static array *state_names* which always has the form of *char* state_names[2]={"h", "v"};*. Once such a line is identified, all relevant information is extracted.

Clearly, a change in the formatting of the source code will inevitably leads to an abstraction of wrong information or none at all. Thus, parsing generated C code in order to gather information about the system should generally be avoided. One feasible solution is to extend the *init file* by the names of the inputs and outputs. Thus all information could be extracted from a single text document. Furthermore it allows changing the values of inputs which are assumed to be zero otherwise. One might also consider to change the formatting of the *init file* into a standardised format like XML. This would enable checking the syntax against a language definition and use readily available parsers to extract all the information.

Another possible solution is to extend the CORBA interface by single or multiple commands that return all system information. Although this is very elegant from a programmer's point of view, it limits the possibilities of usage. It would not be possible to gather system information, if only the code export but no OMC was available.

3.5.2 Using the C Code

Encapsulating the system of differential equations into its own class poses a problem since the C code generated by the OMC makes use of global variables. Namely the structure *DATA*, which contains the values of all the states, algebraic variables and parameters is always referenced via a global variable called *localData*. The usage of global variables, however, has to be avoided when dealing with classes since it causes unwanted interference between different instances of the same class. Consequently, every function has to be converted into a function of the class. It can be achieved by adding a prefix, consisting of the name of the class, to every function definition.

Again, parsing and changing the provided C code is not an elegant solution since changes to the C code might cause this method to fail. In order to avoid altering the C code and to allow the use of a wrapper class, all global variables should be eliminated. If a subroutine needs access to *localData* it should provide a pointer to this structure in its function definition such that the caller is able to pass the structure. The interface of the DASSRT solver for example also features two pointers, namely *rpar* und *ipar*, see Figure 5. They can be used to pass lists of real and integer

```
1  void  DDASRT(
2   int (*res) (..., double *rpar, long* ipar),
3   ...
4   double *rpar,
5   long *ipar,
6   int (*jac) (..., double *rpar, long* ipar),
7   int (*g) (..., double *rpar, long* ipar),
8    ...);
```

Figure 5: Interface Solver DDASRT

parameters to DASSRT. Beside parameters like start time and stop time, the DASSRT interface expects multiple pointers to user functions. These user functions perform the calculation of the residuals, Jacobian matrix or constraints, respectively. All these user functions are called by DASSRT and need access to the information stored in *localData*. The parameter *ipar* could be exploited passing the address to *localData* via a static typecast from *long** to *DATA** and back, see Figure 6. Since both types are pointers no conflicts regarding the size of the variable have to be expected.

Thus it is possible to change all global variables into local ones and thereby to increase the usability of the C code.

```
...
static DATA* localData;
...
int functionDAE_res(..., long int* ipar)
{
  ...

  return 0;
}
```

```
...
int functionDAE_res(..., long int* ipar)
{
  ...
  DATA* localData;
  localData=(DATA*)ipar;
  ...

  return 0;
}
```

Figure 6: local DATA* vs. global DATA*

3.5.3 Implementation

In order to prove feasibility of the suggested solutions, the following changes have been implemented into a local copy of the OMC source code:

1. Extending the *init file* by the names of inputs and outputs
2. Turning localData into a local variable

It has been proved that these changes allow a much more convenient subsequent use of the source code generated by the OMC.

4 PyMbs

A major drawback connected with the usage of OpenModelica is the missing support for the Modelica.Mechanics.Multibody library. Since multibody systems form an essential part in the study of mobile machinery, a tool called PyMbs written in Python has been created at Dresden University of Technology. Using sympy [21], a library for symbolic mathematics, PyMbs generates the equations of motion of arbitrary holonomic multibody systems having the standard form

$$\begin{aligned} \dot{p} &= v \\ M\dot{v} + h &= f + \left(\frac{d\Phi}{dp}\right)^T \lambda \\ \Phi(p) &= 0 \end{aligned}$$

where p is the vector of generalised positions, v the vector of generalised velocities, λ the vector of constraint forces or Lagrange multipliers respectively, M represents the system mass matrix, h is the vector of the gyroscopic and centrifugal forces, f is the vector of all external and elastic forces and Φ contains all holonomic constraints. PyMbs is able to export this system of equations as Modelica code which can then be used within a Modelica model and simulated using OpenModelica. It can also be exported as a MATLAB or a Python file for the use with standard solvers. In order to avoid extremely long equations when calculating the mass matrix M or the vector h explicitly, a recursive formulation [7] exploiting the structure of the multibody system has been implemented. Arising kinematic loops may either be closed by introducing kinematic constraints or using predefined and precalculated kinematic loop objects describing the relation between dependent and independent coordinates. This choice either leads to a DAE or ODE formulation, respectively.

Figure 8 shows an exemplary implementation of a model of a crane crab and a load (see Figure 7). The crab may move horizontally in one axis and the load may rotate around the crab. A force is applied to the crab modelling the effect of a drive. Figure 9 shows the Modelica code, automatically generated by PyMbs. Note, that the model is defined as *partial* since there is no equation defining the magnitude of the driving force. In order to equip PyMbs models with connectors from the Modelica Standard library a new model inheriting all equations from the PyMbs model should be created. Inside the new model mechanical connectors can be instantiated and associated with the corresponding variables as shown in Figure 10.

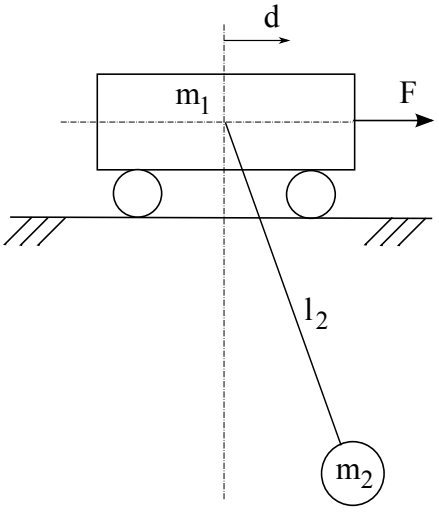

Figure 7: Crane Crab

```
from PyMbs.Input import *

# Set up a new MbsSystem
world=MbsSystem([0,0,-1])

# Define Input and Parameters
F = world.addInput('Force', 'F')
m1 = world.addParam('mass 1', 'm1', 10)
m2 = world.addParam('mass 2', 'm2', 1)
l2 = world.addParam('length', 'l2', 1)
I2 = world.addParam('inertia 2', 'I2', 1/12)

# Define Bodies and Coordinate Systems
crab = world.addBody(name='Crab', mass=m1)
load = world.addBody(name='Load', mass=m2,
                     inertia=diag([0,I2,0]))
load.addCoordSys('joint', p=[l2,0,0])

# Connect Bodies Through Joints
world.addJoint('TransCrab', world, crab, 'Tx',
               startVals=1)
world.addJoint('RotLoad', crab, load.joint, 'Ry')

# Add Sensors and Force Elements
world.addLoad('DrivingForce', 'PtPForce',
              crab, world, F)
world.addSensor('Position', 'Distance',
                crab, world, 'd')

# Calculate Equations of Motion and Generate Code
genEquations(world, explicit=True)
genCode('mo', 'CraneCrab_PyMbs')
```

Figure 8: PyMbs Source Code of a Crane Crab

```
// This file was generated by PyMbs
partial model CraneCrab_PyMbs
        // Positions
        Real[2] q (start={1,0})
                "q_TransCrab,q_RotLoad";
        // Velocities
        Real[2] qd (start={0,0})
                "qd_TransCrab,qd_RotLoad";
// Inputs
        Real F;
// Parameters
        parameter Real I2 = 0.083 "inertia_2";
        parameter Real g  = 9.81  "gravity";
        parameter Real l2 = 1     "length";
        parameter Real m2 = 1     "mass_2";
        parameter Real m1 = 10    "mass_1";
// Sensors
        Real[2] d;
// Variables
protected
        Real[2]   WF_DrivingForce;
        Real[2,2] M;
        Real[2]   h;
        Real[2]   f_gravity;
        Real[2]   f_ext;
        Real[2]   f;
equation
        der(q) = qd;

        d = {abs(q[1]),q[1]*qd[1]/abs(q[1])};

        WF_DrivingForce = {q[1]/abs(q[1]),0};

        M = {{m1+m2,l2*m2*sin(q[2])},
             {l2*m2*sin(q[2]),I2+m2*l2^2}};

        h = {l2*m2*qd[2]^2*cos(q[2]),0};

        f_gravity = {0,-g*l2*m2*cos(q[2])};

        f_ext = F*WF_DrivingForce;

        f = f_ext+f_gravity;

        M*der(qd) = f - h;

end CraneCrab_PyMbs;
```

Figure 9: PyMbs Modelica Output

With only few enhancements to the model of the crane crab, PyMbs is able to generate an interactive graphical output(see Figure 11). It enables the user to check the consistency of the model by manipulating the generalised coordinates via sliders. The effect on the multibody system can be evaluated ad hoc.

In case the available collection of joints, force elements and sensors do not suffice, PyMbs can be extended very easily due to its object oriented structure. Moreover, it takes only very little effort to implement further output formats.

PyMbs is freely available. For further information please contact one of the authors.

```
model CraneCrab
  extends CraneCrab_PyMbs;
  import Modelica.Mechanics.Translational.*;
  // Mechanical Connector
  Interfaces.Flange_b flange;
equation
  flange.s = d[1];
  flange.f = F;
end CraneCrab;
```

Figure 10: Usage of PyMbs Output in Modelica

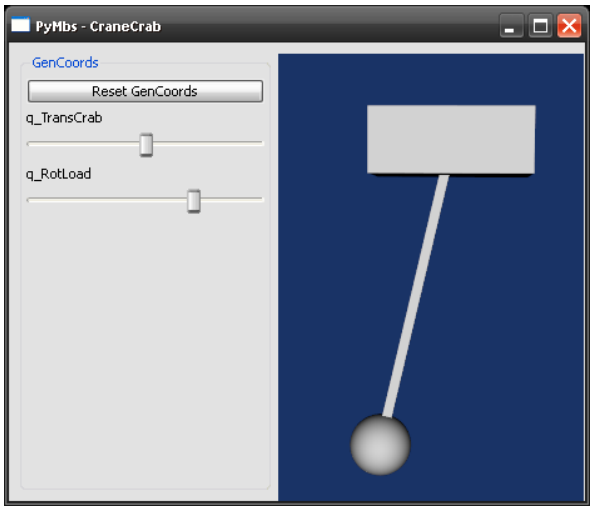

Figure 11: Graphical PyMbs Output of the Crane Crab

5 Example Models

Based on the described tool chain, several models have already been implemented. In this paper, a wheel loader shall be presented (Figure 12 and 13). The purpose of the wheel loader model is the assessment of innovative operational controls and novel assistance systems. A realistic driving experience is achieved by connecting the model

via SARTURIS to a motion platform (Figure 1). In order to provide the user with his familiar operating environment, the manufacturer provided a real driving cab, which has been installed onto the motion platform and is integrated via CAN Bus (Figure 14).

Figure 12: Virtual Reality Simulation of a Wheel Loader (Exterior View)

Figure 13: Virtual Reality Simulation of a Wheel Loader (Operator View)

The mechanics of this model have been described using PyMbs, exported to Modelica and equipped with mechanical connectors. Hydraulics, drivetrain as well as control systems and a tire model [23] have been modelled directly within Modelica. This model was then translated into a SARTURIS module and integrated into the simulation environment. It is now possible to run a model of the wheel loader on the motion platform which is equipped with the control units (pedals, joystick, steering wheel...) from the real machine. They allow interaction with the simulation through an operator in real time.

Figure 14: Changing the Cabin of the Motion Platform

6 Future Work

Increasing pressure on costs and time foster a need for a more efficient design process. Prior research has facilitated simulation processes that enable companies to shorten the design process by using virtual prototypes. However, these methods are hardly used in the branch of mobile machinery. Machines comprise numerous components manufactured by different companies. Thus, successful simulation requires a cooperation of the manufacturer and his suppliers. Due to the risks connected with transferring crucial information a cooperative simulation process has not yet been established in industry. INPROVY[2] [20], a research project coordinated by the Dresden University of Technology, aims at overcoming those obstacles by providing methods that allow simulations across company borders. Furthermore, our research focuses on the reuse of available information and its administration as well as its protection.

[2]BMBF support code: 02PC1110

7 Conclusion

It was shown that it is possible to conduct real time simulations in a virtual reality environment with Modelica by using our simulation framework SARTURIS. A tool called OpenModelicaToSarturis has been developed which automatically converts Modelica models into SARTURIS modules using OpenModelica. The lacking support of the Modelica.Mechanics.Multibody library by OpenModelica has been overcome by using PyMbs. PyMbs is a tool, developed in Python, which allows modelling of holonomic multibody systems. It offers different output formats like Modelica, MATLAB and Python code. The presented methods and tools are very beneficial to support the modelling and use of interactive VR technologies.

Dresden University is very interested in cooperation with other universities which might want to use SARTURIS for their research.

References

[1] Koo, T. Y.; Bae, C. H.; Kim, B. Y.; Rowland, Z.; Suh, M. W.: Development of a driving simulator for telematics human-machine interface studies.-Proceedings of the Institution of Mechanical Engineers, Part D (Journal of Automobile Engineering) * Band 222 (2008) Heft 11

[2] Zschocke, A. K.; Albers, A.: A method to examine links between subjective and objective evaluations of steering torque utilising a model-based approach.-FISITA, World Automotive Congress, 32 * (2008)

[3] Pasetto, M.; Gamberini, L.; Manganaro, A.: Potential of immersive virtual reality models in studies of drivers' behaviour and interventions to improve road safety.-PRESENCE, Annual International Workshop on Presence, 11 * (2008)

[4] VTT Technical Research Centre of Finland: HumanICT - New Human-Centred Design Method and Virtual Environments in the Design of Vehicular Working Machine Interfaces VTT Working Papers.-ISBN-Nr.: 978-951-38-6625-9

[5] Strassburger, S; Schulze, T.; Fujimoto, R.: Future trends in distributed simulation and distributed virtual environments.-WSC, Winter Simulation Conference, 40 * (2008)

[6] Beater, P.; Otter, M.: Multi-Domain Simulation: Mechanics and Hydraulics of an Excavator. In: Proceedings of Modelica 2003 conference, 2003

[7] Fisette, P.; Samin, J. C.: Symbolic generation of large multibody system dynamic equations using a new semi-explicit Newton/Euler recursive scheme. Archive of Applied Mechanics, Vol. 66, Issue 3, pp. 187-199 (1996)

[8] Fritzson, P.: Principles of Object-Oriented Modeling and Simulation with Modelica2.1.Wiley-IEEE Press, 1 2004.

[9] Fritzson, P.; et al.: OpenModelica System Documentation.www.openmodelica.org, 1 2008.

[10] Penndorf, T.; Kunze, G.: Codegenerator fuer die Echtzeitsimulation von Mehrkoerpersystemen.-ASIM 2006 19. Symposium Simulationstechnik, Universitaet Hannover, September 2006

[11] Penndorf, T.: Universelles Framework zur Abbildung von Maschinenmodellen in virtuellen Umgebungen. In: Schriftenreihe der Forschungsvereinigung Bau- und Baustoffmaschinen (2006) 34

[12] Penndorf, T.; Kunze, G.: "Durchgespielt"-Interaktive Simulation von Baumaschinen. IX-MAGAZIN FUER PROFESSIONELLE INFORMATIONSTECHNIK Heft 08/2007.-Heise Zeitschriften Verlag, Hannover

[13] OpenSceneGraph, http://www.openscenegraph.org/

[14] GTK, http://www.gtk.org/

[15] CAN in Automation e. V.. http://www.can-cia.org

[16] SUNDIALS (SUite of Nonlinear and DIfferential/ALgebraic equation Solvers) http://www.llnl.gov/CASC/sundials/

[17] CMake, http://www.cmake.org/

[18] Frenkel, Jens: Integration von OpenModelica in das Programmsystem SARTURIS.-TU

Dresden, Professur fuer Baumaschinen- und Foerdertechnik, Diplomarbeit,Februar 2009

[19] Schubert, C.; Frenkel, J.: PyMbs Userguide.- TU Dresden, Professur fuer Baumaschinen- und Foerdertechnik, Forschungsbericht, September 2009

[20] www.inprovy.de

[21] sympy, http://code.google.com/p/sympy/

[22] OpenModelica, http://www.openmodelica.org/

[23] Zimmer, Dirk and Otter, Martin(2009)'Real-time models for wheels and tyres in an object-oriented modelling framework',Vehicle System Dynamics,99999:1,

