Interactive FMU-based Visualization for an Early Design Experience

Volker Waurich[1] Jürgen Weber[2]

[1]Chair of Construction Machines, TU Dresden, Germany, volker.waurich@tu-dresden.de
[2]Chair of Fluid-Mechatronic Systems, TU Dresden, Germany, weber@ifd.tu-dresden.de

Abstract

User experience is an eminent part of holistic product design. Especially in the field of mobile machinery, the driver's impression of the machine handling is crucial for successful design. To get an early understanding of the ergonomic aspects of a new concept of operation, functional prototypes can be applied. This paper presents the tools to develop a functional prototype using free software and low-cost hardware. This includes prototyping of control devices, interfaces to the Modelica-based simulation models and a generic visualization using a game engine. In order to speed up the process of functional prototyping, an approach to automatically visualizing FMUs based on a scene description file is presented. The application of interactive simulation was used to support the development of a novel control device for excavators in a student project at TU Dresden.
Keywords: visualization, OpenModelica, engineering education, construction machines, rapid prototyping

1 Introduction

The operation of mobile machinery, e.g. excavators, puts ambitious requirements on the driver. Therefore, the ergonomic aspects of the control environment are an important selling point. The innovation of new operating concepts should be supported by an early design experience. In a student project at Technische Universität Dresden, a collaboration of students from the fields of Technical Design, Mechanical Engineering and Media Computer Science developed an innovative control concept for mobile excavators. The project was initiated by an OEM of mobile machinery. Although the actual project results are confidential, the applied methods and tools shall be presented and serve as a motivation for similar projects.

To support the design process, a prototypic control device was engineered to get a haptic experience. With the help of novel rapid prototyping technologies, as *3d-printing* or *lasercutting*, complex designs can be realized quickly and cheaply. Since the required machines became affordable, public workspaces, so-called *makerspaces* spread out more and more. Due to that, even with a small budget, realistic prototyping is possible. Another innovation is the availability of easy-to-use, low-cost microcontrollers. Using different sensors, e.g. potentiometers, motion concepts of the prototypes can be tested. Utilizing functional prototypes during an early design phase, facilitates more profound impressions of the product than using CAD-models or plastic prototypes. Machine tools and electronics are available in Makerspaces and easy to apply for students. With an easy-to-use connection to virtual environments based on simulation models, the design process can be enhanced further. With the help of the OpenModelica tool chain, an FMU-visualization has been developed which allows an automated generation of appealing 3d environments.

This paper covers the different aspects of developing functional prototypes with a high level of automation and tool support. For the presented use-case of the machine control development, only freely available software (i.e. open-source Modelica-tool *OpenModelica* and the free gaming engine *unity*), low-price hardware and cheap prototyping technologies that are becoming widely accessible, are applied. This paper is meant to be a motivation for combining physical prototyping and virtual mockups within the training of engineers. As experience has shown, the development of functional prototypes creates a high level of self-motivation and perfectionism among participants.

In chapter 2, the applied methods of physical rapid prototyping are presented. Chapter 3 discusses the means of developing interactive Modelica models. Afterwards, the basic idea behind a generic FMU-visualization is presented and the tools for visualizing the simulation models are introduced in chapter 4. The presented approach is compared to existing visualization workflows. Chapter 5 describes the manufacturing of a control device. Finally, chapter 6 concludes the paper and gives an outlook on future work.

2 Physical Prototype Manufacturing

2.1 Makerspaces for Higher Education

In recent years, affordable technologies for rapid prototyping have spread widely. Various libraries and higher educational institutions like Saxon State and University

Library Dresden (slu) offer public access to rapid prototyping machines in so called makerspaces. Makerspaces are collaborative work spaces which provide rapid prototyping tools and the knowledge to utilize them. Typically, projects in the field of model making and electronics can be realized using the facilities of a makerspace. The interest in makerspaces emerges and despite the lack of long-term investigation, the impact on engineering education has been promising among many universities (ISA, 2016). Ongoing studies will give an overview of how the overall impact of academic makerspaces has to be assessed. At least with regard of the presented student project, a high level of motivation to realize the projects and to acquire the necessary knowledge has been observed.

In order to develop a control device for an excavator, 3D printers and foam cutters have been used to produce haptic prototypes. The production costs are very low and therefore are best suited to use them in student projects. Project participants from the field of technical design developed design drafts which have been modelled in CAD-software. The printed 3D prototypes give a spatial impression and provide enough stability to integrate joints and sensors.

2.2 Application of Sensors and Microcontrollers

Besides machine tools, makerspaces offer a range of electronic components and easy-to-use microcontrollers such as Arduino (Ard). With these low-cost controllers, sensor concepts can be set up easily and data can be processed and transferred to a computer. In the presented project, buttons, rotary and translational potentiometers have been set up to map the functionalities of a conventional excavator control. The sensors have been attached in the joints of 3D-printed control devices in order to access the control device condition. The Arduino reads the sensors and transfers the signals to a computer, either via USB-connection or with an additional Bluetooth module. The messages can be processed by *SerialPortReceive* of the *Modelica_DeviceDrivers* library. When using the Arduino IDE, users write C-like code, compile and transfer it directly to the board and are able to monitor serial connection communication. There is a vast amount of documentation and tutorials available that simplifies microcontroller programming for students outside this subjects area. The following Arduino code can be applied to transfer signal data of the Arduino's analog pin 1 via USB-connection with a sample time of 0.1 s.

```
byte buf[2];
unsigned long lastSignal = 0;
unsigned long interval = 100; //ms
int value = 0;

void setup() {
  Serial.begin(9600);
}

void loop() {
while(millis() - lastSignal > interval)
{
  lastSignal += interval;
  value = analogRead(1);
  buf[0] = lowByte(value);
  buf[1] = highByte(value);
  Serial.write(buf,2);
}}
```

3 Modelling of Interactive Simulation Environments

3.1 Model Interaction

The use-case of models involving external inputs during runtime became much more accessible by *Modelica_DeviceDrivers* library (*M_DD*) (Bellmann, 2009). The library interfaces various input devices and communication protocols. Hence, Modelica models can be enhanced with direct user-inputs or connected to other processes during runtime. A realtime synchronization is provided as well. For the presented demonstrator, the serial port implementation was utilized in order to communicate with an Arduino microcontroller. *M_DD* supports packing and unpacking of byte-messages which allows to access data e.g. sensor signals via a serial port connection. *OpenModelica* supports serial communication and packaging both in simulation mode and in *FMU*s. Figure 1 displays the graphical model view of an excavator model, that is controlled via serial communication. The message protocol is modelled with unpackInt-models, that split incoming messages into a sequence of integer variables. These integer variables are converted to real variables and conditioned to fit the excavator interface. The excavator model has been taken from a Modelica-library by the Chair of Construction Machines, TU Dresden.

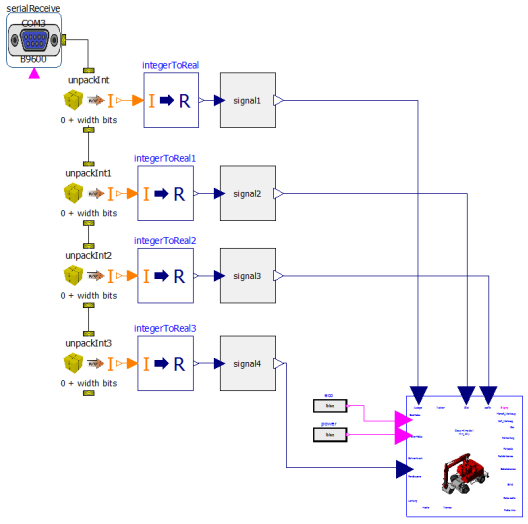

Figure 1. Model of an excavator and a serial port interface using *Modelica_DeviceDrivers* library

DOI 10.3384/ecp17132879

The following listing shows the parametrization of a model to read a two-byte message sent by an Arduino, whereas the parameters *baud*, *sampleTime userBufferSize* and *Serial_Port* have to be adapted to the sending controller. The *width* parameter of the *UnpackUnsignedInteger* model has to be set to 16 bit in order to deserialize the two byte value, sent from the microcontroller.

```
model arduino
Modelica_DeviceDrivers.Blocks.Communication
.SerialPortReceive
arduinoRead(
baud=Modelica_DeviceDrivers.Utilities
.Types.SerialBaudRate.B9600,
parity=0,
enableExternalTrigger=false,
startTime=0.0,
autoBufferSize=false,
userBufferSize=2,
sampleTime=0.1,
Serial_Port="COM5");

Modelica_DeviceDrivers.Blocks.Packaging
.SerialPackager.UnpackUnsignedInteger
unpackInt(
bitOffset=0,
width=16,
nu=1);
equation
  connect(arduinoRead.pkgOut,
      unpackInt.pkgIn);
end arduino;
```

3.2 Realtime Capabilities

Realtime requirements restrict the model to simulating within a specified interval of time. Hard realtime criteria demand a deterministic execution time whereas soft realtime allows the simulation to exceed the time limit occasionally. In the presented use case, soft realtime criteria are assessed. Nevertheless, for realtime application, it is favourable to reduce the simulation time. Modelica compilers allow different kinds of performance optimization for simulations. The time integration method has a big influence on the execution time, depending on the number of iterations and step sizes. In most realtime applications, explicit, fixed step methods are preferable. The lack of stability and the necessity of small step sizes lead to the development of more sophisticated methods e.g. inline integration (Elmqvist et al., 1995). Besides that, the evaluation of parameters is an effective option to increase simulation speed. There are various optimization techniques to improve calculation of algebraic loops, e.g. structural methods like tearing (Elmqvist and Otter, 1994) or reshuffling (Waurich et al., 2014). Calculations of jacobi matrices can perform differently dependenig on whether numerical, symbolical or colored jacobians are used. Automatic parallelization is also a feature to speed up simulation. Of course, the operating system, the hardware and the C/C++ compiler influence the simulation time as well.

The excavator model consists of a multi body system and some simple hydraulic components (cylinders, valves, flow sources). In most cases, the BLT-matrix of a mechanical model is dominated by a linear system of equations. Hence, parallelization of BLT-blocks will not improve the simulation speed. The present model benefits mostly from the evaluation of parameters. The dominating system of equations with 437 equations including 9 tearing variables is reduced to a system with 379 equations including 7 tearing variables. The amount of single equations reduces from 1208 to 1162. This results in a simulation speed-up of 1.33. This is sufficient to run the simulation without exceeding the realtime limits on a Windows 7 desktop computer with i7-3930K processor. The *FMU* was compiled using OpenModelica and gcc 5.3.0 as *FMU 2.0 model exchange*.

4 A Generic Visualization of FMUs

4.1 The Functional Mock-Up Unit

In order to exchange simulation models and to use them across various software, the Functional Mock-Up Interface was developed (Blochwitz et al., 2012). The *Modelica* language and its tools are highly involved in the development and application of *FMI*. The *FMI*-standard features two variants, i.e. *model-exchange* without internal time integration and *co-simulation* that includes a time integration solver. The black-box models that provide the FMI-API are called *Functional Mock-Up Units* and contain the functional behaviour of a simulation model that can be accessed via interface variables. The model variables are listed in the `modelDescription.xml`. The connections and relations of these model variables are hidden from the user since *FMUs* are compiled as a shared library. This is very useful since it protects intellectual property but it is cumbersome if information of the model structure is of interest. Hence, a generic visualization of *FMUs* is not possible in general.

4.2 Existing Approaches to Visualize Multibody Models

Commercial *Modelica*-tools offer built-in visualization features for multibody systems based on the *Modelica.Mechanics.MultiBody.Visualization.Advanced* models. Visualization comprises both subsequent and concurrent visualization of simulation. This visualization is possible since the tools have full access to the model information and the variables that are used to visualize the shapes. Another approach would be to add dedicated animation objects to the Modelica model and let them communicate with an external visualization software, e.g. in the commercial *DLR Visualization library* (Hellerer et al., 2014). Also the *Modelica3D* implementation by Höger relied on Client/Server communication (Hoeger et al., 2012). Yamaura et al. (Yamaura et al., 2016) described a comprehensive framework of different tools that exchange model variables via *UDP* communication with

a corresponding *Unity* model. This approach combines the physical model capabilities of engineering tools like *Simulink* and *Dymola* with the highly developed gaming engine *Unity* which offers much more visualization and graphical modeling features than any simulation software. Another promising implementation for discrete time simulations was presented by (Bijl and Boer, 2011) , that is designed on a database which feeds the 3D visualization. The use of appealing 3D visualization and the potential of 3D game engines is described as well. An entirely different concept was presented in (Elmqvist et al., 2015) in which even the modeling is performed in a 3D visualization environment that provides direct feedback on the model structure of a multibody system. This visualization uses the web interface of the simulation tool *Dymola*.

Since there was no free Modelica tool that features visualization in an integrated manner, the open-source Modelica Compiler *OpenModelica* and its graphical editor *OMEdit* have been enhanced to visualize results of simulations. Therefore, the OpenModelica Compiler has to extract all necessary information about the visualization shapes from its internal model representation. Hence, the animation of *Modelica.Mechanics.MultiBody* models can be provided without adding dedicated visualization objects to the model.
Instead of implementing a new OpenModelica-specific API to transfer visualization variables between simulation and animation-software, the authors decided to choose an already existing API, i.e. the *FMI*. By means of a visualization scene description file that is generated by the *OpenModelica Compiler*, the visualization software in *OMEdit* can access relevant variables and maps them to the corresponding animation shape properties. In the following chapters, the details of FMU-based visualization are presented.

4.3 A Specification of Visualization

As described in the previous chapter, *OpenModelica 1.11* is able to create a scene description XML-file that contains the information about the *Modelica.Mechanics.MultiBody.Visualizers.Advanced.Shape* objects within a model. The *shape* model contains the basic visualization information like position, orientation, scale and color. This approach was already mentioned in (Waurich et al., 2016) and a proof of concept implementation was presented. The scene description XML-file simply lists all instances of the *Shape* model and assigns values to their parameters. The following exemplary snippet of a scene description XML-file contains information about the model "shape1" which is of `type` "cylinder". The position vector `r` is defined by constant expressions and lies in the root "{0,0,0}" whereas the `length` attribute depends on the component reference "shape1.length".

```
<visualization>
  <shape>
    <ident>shape1</ident>
    <type>cylinder</type>
    <r><exp>0.0</exp>
       <exp>0.0</exp>
       <exp>0.0</exp>
    </r>
    <length>
       <cref>shape1.length</cref>
     </length>
  </shape>
</visualization>
```

The shape parameters are either defined by an `<exp>` tag which refers to a constant expression of type real or to a `<cref>` tag, which stands for a reference given by a string-type. `<cref>` elements have to be updated during runtime. Shapes can be either geometric primitives or CAD-files, like .stl or .dxf that are referenced by their absolute path names in the scene description file. Besides the `shape` models, there are more visualization models that could be defined, e.g. `Surface` or `PipeWithScalarField`, but the current implementation covers `shape` only. A *XML Schema Definition* is available at `https://github.com/vwaurich/visxml`

4.4 The Visualization Architecture

No matter which frontend is used to display the 3D scene, the mechanism to animate the shapes is identical as depicted in Figure 2. The visualization backend needs an *FMU* and a corresponding scene-description file, both generated by the *OpenModelica Compiler*. It has to be ensured, that all variables which are used to visualize the scene, are accessible in the *FMU*. This means that these variables must be retrievable via *fmiGetReal* API. Therefore, *OpenModelica* changes protected variables to public if needed.

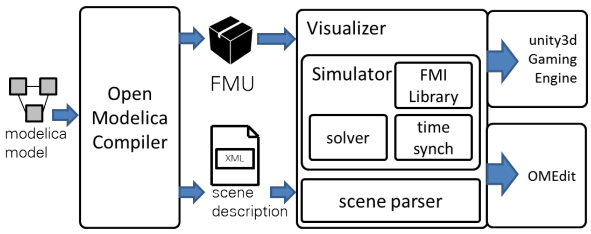

Figure 2. Overview of *FMU*-based visualization both with *unity* and *OMEdit* frontend.

After the selection of an *FMU*, the visualization backend instantiates all shapes listed in the scene description file. These can be either geometric primitives such as cubes or spheres, or imported CAD-files. Constant shape properties can be set directly during initialization of shapes. In constrast, variable properties cannot be set before the solution of the inital system of the *FMU*.

Unpacking, loading, instantiation, initialization and simulation of the *FMU* is performed by the *FMILibary*

DOI 10.3384/ecp17132879

(FMI). For the *Model Exchange FMU*s, a simple Explicit Euler solver with a default step size of 1ms is used. The simulation is synchronized with realtime by the visualizer backend itself. Hence, no synchronization on the model side is necessary (e.g. *from Modelica_DeviceDrivers Blocks.OperatingSystem.SynchronizedRealtime*).

Compared to other visualization approaches, the generic FMU visualization has the following advantages:

- A specification of the visualization objects allows different tools to create the same scene automatically.
- No model modifications have to be applied in order to generate a visualization. No additional dependencies have to be included. No additional equations are added to the existing multi-body model.
- It is easy for simulation tools to generate scene description files. Based on this visualization formalism, the visualization is independent of the simulation software and does not rely on vendor specific interfaces.
- It enables automatic integration of physical simulation in graphical modelling software (as will be shown for the gaming engine *unity*).
- The simulation and variable access is achieved via shared memory communication and therefore does not need (but can be extended for) simulation via a network connection.
- It is helpful to visualize third party FMUs automatically to get an understanding of their behaviour without having access to the model itself.
- It is more convenient to add and edit advanced visualization features in a proper visualization tool and not in the simulation model by a Modelica-Editor.

4.5 OMEdit FMU-Visualization

The graphical connection editor *OMEdit* features basically textual and graphical modeling views, result plotting and algorithmic debugging. The lack of 3D animation hindered the use for mechanical applications. The novel implementation of a result-file based and FMU-based visualization helps to get a better understanding of mechanical systems.

Figure 3 displays the visualization view of *OMEdit*. The visualization is implemented using OpenSceneGraph and features the animation of mat-result files, csv-result files and *FMU*s. In each case, a scene description file is needed, to map the model variables to the shape properties.

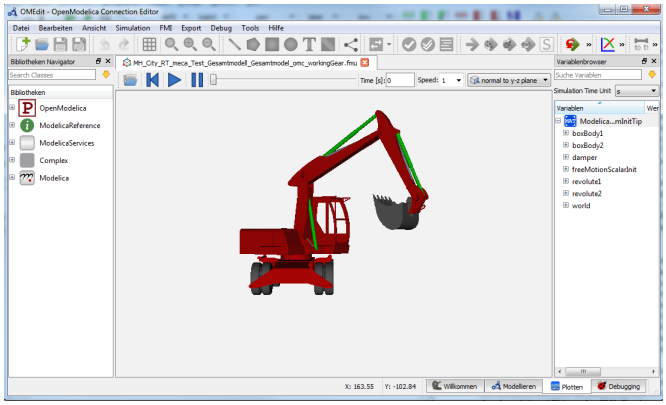

Figure 3. Screenshot of the visualization perspective in *OMEdit*.

4.6 Unity FMU-Visualization

The implementation in *OMEdit* based on OpenSceneGraph is not visually attractive and makes it very cumbersome to enhance the scene with additional graphical objects. A gaming engine with graphical editor and a huge asset store like *unity* (Uni), would allow an easy setup of appealing graphical scenes as in Figure 4. Hence, the mechanism of loading an *FMU* and a scene description file has been implemented in a *unity* plugin. The user simply chooses an *FMU* via a dialog and the plugin creates so called *GameObjects* for the corresponding shapes. Besides that, an *FMU*-simulator GameObject is created, which simulates the FMU and accesses the necessary variables. This comprises everything to run the scene either in the unity debugger or from a compiled unity project.

Figure 4. Unity scene with an FMU-based excavator model that is controlled by an Arduino board in realtime.

Next to the shape objects and the FMU-simulator, additional GameObjects can be added in order to create an adequate environment. Accessing the FMU-inputs and FMU-outputs from the unity model is possible via interface functions of the FMU-simulator GameObject. Hence, the FMU-generating simulation tool is only responsible for the physcial simulation. The graphical modelling can be performed by a special purpose tool. The FMU-simulator

plugin supports this separation by generating the basic mapping between simulation and visualization automatically. The unity user interface with FMU-selection dialog and loaded FMU-visualization is depicted in Figure 5. Since the FMU has to be initialized to calculate the position, orientation and color of the bodies, all shapes are still in the root of the coordinate system.

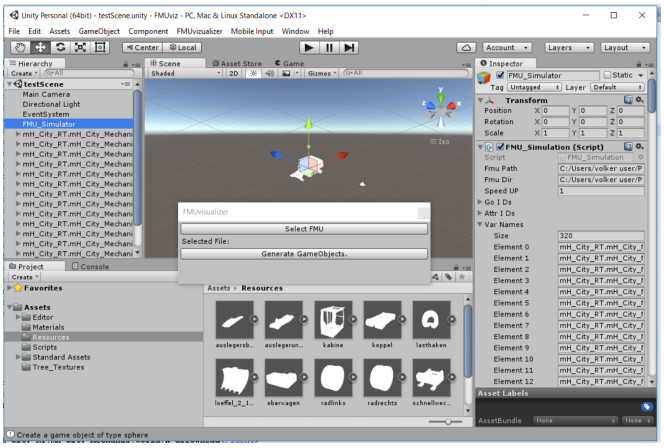

Figure 5. Unity user interface with loaded FMU. The GameObjects for the shapes and the FMU-Simulator are listed in the hierarchy view, the .dae files are copied to the resources and the inspector view displays all variables that are updated during runtime.

When interchanging variables between the unity world and the Modelica-based FMU, it has to be considered, that the coordinate systems are different. Modelica uses a right-handed system whereas unity relies on a left-handed system. Furthermore, the y-axis should be used as vertical since *unity* uses it as vertical by default (which is essential since available skyboxes display a horizon in the x-z-plane).

The FMU-simulator plugin automatically converts the position and orientation of the Modelica-variables to the left-handed system of the unity variables and switches the vertical axis if desired. Another issue is the lack of stl-file support in unity. It needs an stl-importer plugin or the CAD-files have to be converted to a 3D data format e.g. COLLADA. File conversion can be done manually or scripted by tools like blender (Ble).

5 The Development of a Remote Control Device for an Excavator

The previous chapters depicted the necessary tools to set up a functional prototype. To try out novel control concepts, physical prototypes have been equipped with sensors to measure the motion of the joints. The signals are used to control the volume flow in and out of the cylinders. Hence, the velocity of motion for the boom, the arm and the shovel are controlled. Even inverse kinematics can be tried out if the cylinders are controlled to follow cartesian inputs to set the position of the shovel. Furthermore, assistance systems are experienceable without implementing them in fully operable systems. This simplifies the evaluation of acceptance and ergonomics. Even exceptional control mechanisms like handheld controllers for remote control are possible. Figure 6 shows the setup to control a model in OMEdit via Bluetooth connection, which is handled as an ordinary serial port.

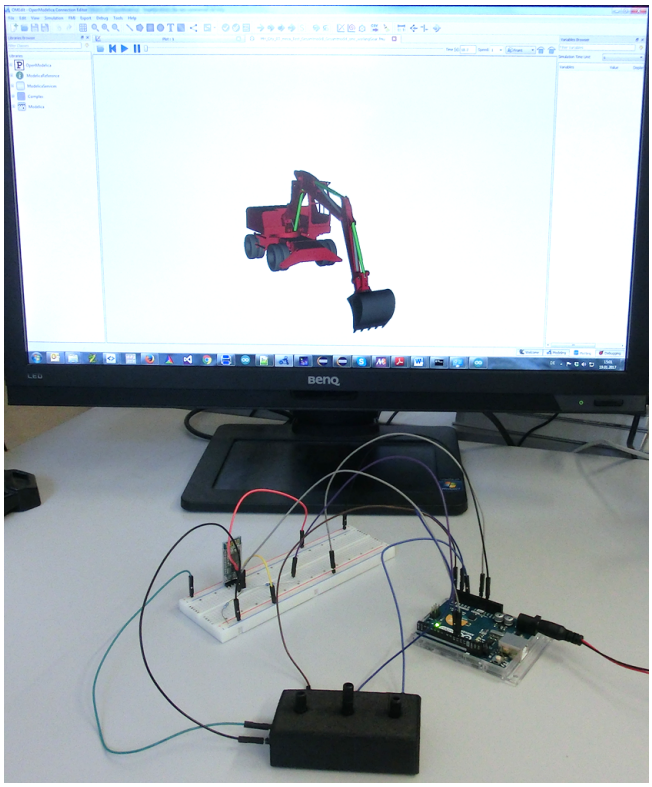

Figure 6. Remote control setup to control an excavator model in OMEdit via Bluetooth connection. The control device is a printed box with 3 rotary potentiometers.

The unity editor allows further settings for camera position (first or third person view) as well as lighting and terrain modelling. In the unity asset store, various objects to populate the scene can be downloaded for free or charged.

6 Conclusion and Outlook

This paper comprises a workflow for developing functional prototypes that have been realized within a student project at TU Dresden. The usage of Makerspace facilities, low-budget electronics and free software together in an interdisciplinary design project, was a successful experiment. The motivation of students was huge and both the familiarisation with novel technologies as well as its application are valuable experiences. Besides the individual learning success, the developed prototypes are highly praised by the project initiator, an OEM of excavators.

During the project, improvement opportunities have been revealed. Basically, the development of a virtual environment which can be controlled with external

DOI 10.3384/ecp17132879

hardware in realtime and based on the simulation of a Modelica model needed improvement. Hence, an automated approach to setup visualizations of FMU-based multi-body systems was implemented. The integration via FMUs in the game engine *unity* leads to satisfying results. More importantly, the scene description of FMUs enables new generic interfaces to visualization tools and their features. As a future extension, Modelica models could be extended with contact-force-interfaces or collision interfaces that could be generated automatically in a unity project in order to interact with *unity*s physics engine and feedback the results to the simulation model. In the field of mobile machinery, interaction to soil or particle models in unity would be very useful as well. Since Game Engines feature comprehensive possibilities to model environments, experimental grounds can be set up to test e.g. assistance and automation systems. Through the network protocol interfaces of *M_DD*, even web-based services in mobile machines can be experienceable in early design stages.

The scene description file is currently an OpenModelica specific feature. Further support of this FMU extension would leverage the advantage of the unity-plugin and the development of other FMU-Visualization-Add-Ons in additional tools. A discussion about adding the scene description file as an optional extension to the FMI-Standard would be highly appreciated by the authors.

References

The arduino webpage. `www.arduino.cc`. Accessed: 2016-11-18.

The blender webpage. `www.blender.org`. Accessed: 2016-12-08.

The fmilibrary webpage. `www.jmodelica.org/FMILibrary`. Accessed: 2016-11-21.

The unity3d webpage. `www.unity3d.com`. Accessed: 2016-11-18.

The saxon state and university library dresden (slub) webpage. `http://www.slub-dresden.de/en/service/workplaces-workspace/makerspace/`. Accessed: 2016-12-07.

Proceedings of the 1st International Symposium on Academic Makerspaces ISAM 2016, 2016. URL `www.project-manus.mit.edu/home/conference`.

Tobias Bellmann. Interactive simulations and advanced visualization with modelica. In *Proceedings 7th Modelica Conference*. LinkÃűping University Electronic Press, 2009.

Jonatan L. Bijl and Csaba A. Boer. Advanced 3d visualization for simulation using game technology. In *Proceedings of the Winter Simulation Conference*, WSC '11, pages 2815–2826. Winter Simulation Conference, 2011. URL `http://dl.acm.org/citation.cfm?id=2431518.2431853`.

Torsten Blochwitz, Martin Otter, Johan Åkesson, Martin Arnold, Christoph Clauss, Hilding Elmqvist, Markus Friedrich, Andreas Junghanns, Jakob Mauss, Dietmar Neumerkel, Hans Olsson, and Antoine Viel. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. pages 173–184, 2012. doi:10.3384/ecp12076173.

Hilding Elmqvist and Martin Otter. Methods for tearing systems of equations in object oriented modeling. In *In ESM 94 European Simulation Multiconference*, 1994.

Hilding Elmqvist, Martin Otter, and Franşois E. Cellier. Inline integration: A new mixed symbolicnumeric approach for solving differential-algebraic equation systems. In *Proceedings of the 1995 European Simulation Multiconference*, pages 23–34. Society for Computer Simulation International, June 1995.

Hilding Elmqvist, Alexander D. Baldwin, and Simon Dahlberg. 3d schematics of modelica models and gamification. In *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*, number 118, pages 527–536. Linköping University Electronic Press, Linköpings universitet, 2015.

Matthias Hellerer, Tobias Bellmann, and Florian Schlegel. The dlr visualization library - recent development and applications. In *Proceedings of the 10th International Modelica Conference; March 10-12; 2014; Lund; Sweden*, number 96, pages 899–911. Linköping University Electronic Press; Linköpings universitet, 2014. doi:10.3384/ecp14096899.

Christoph Hoeger, Alexandra Mehlhase, Christoph Nytsch-Geusen, Karsten Isakovic, and Rick Kubiak. Modelica3d - platform independent simulation visualization. In *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*, number 76, pages 485–494. Linköping University Electronic Press; Linköpings universitet, 2012. doi:10.3384/ecp12076485.

Volker Waurich, Ines Gubsch, Christian Schubert, and Marcus Walther. Reshuffling: A symbolic pre-processing algorithm for improved robustness, performance and parallelization for the simulation of differential algebraic equations. In *Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, EOOLT '14, pages 3–10, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2953-8. doi:10.1145/2666202.2666203. URL `http://doi.acm.org/10.1145/2666202.2666203`.

Volker Waurich, Martin Großer, and Sebastian Voigt. Generische visualisierung von fmu-basierten modellen für die interaktive simulation. In *Tagungsband Workshop ASIM STS/GMMS 2016*, pages 230–236. ASIM STS/GMMS, 2016. ISBN 978-3-901608-48-3.

Masahiro Yamaura, Nikos Arechiga, Shinichi Shiraishi, Scott Eisele, Joseph Hite, Sandeep Neema, Jason Scott, and Theodore Bapty. Adas virtual prototyping using modelica and unity co-simulation via openmeta. In *The First Japanese Modelica Conferences, May 23-24, Tokyo, Japan*, number 124, pages 43–49. Linköping University Electronic Press, Linköpings universitet, 2016.