Modelica on the Web

Tamas Kecskes[1] Patrik Meijer[1] Janos Sztipanovits[1] Peter Fritzson[2] Adrian Pop[2] Arunkumar Palanisamy[2]

[1]Institute for Software Integrated Systems, Vanderbilt University, USA,
{tamas.kecskes,patrik.meijer,janos.sztipanovits}@vanderbilt.edu
[2]PELAB - Programming Environment Lab, Dept. of Computer and Information Science, Linköping University, SE-581 83 Linköping, Sweden, {peter.fritzon,adrian.pop,arunkumar.palanisamy}@liu.se

Abstract

Modelica has been around as a language from the late 1990's and since then a range of compilers and editors have emerged. Currently none of these environments provide a web-based user interface and follow the approach of requiring each end-user to install the application (typically together with a set of dependencies) on their local machine. This in itself may or may not be of major concern. Of more importance is their current lack of a seamless collaborative approach to modeling. This paper presents the first web-based collaborative graphical and textual modeling environment for Modelica based on WebGME and OpenModelica. Graphical composition of Modelica models from component libraries is supported via WebGME. Textual editing of the composite model is possible via OMWebBook.

Keywords: web-based modeling, collaborative modeling, metamodeling, WebGME, OpenModelica

1 Introduction

In the first part of this paper a web-based, online, collaborative Modelica (Fritzson and Engelson, 1998; Fritzson, 2014) modeling environment will be presented. It builds on WebGME (Generic Modeling Environment) and currently existing Modelica compilers, specifically OpenModelica and JModelica.org. This paper will demonstrate how WebGME, and to a certain degree metamodeling in general, can be used to rapidly create tailored modeling environments. The environment already includes key aspects such as collaboration, centralization of storage and distribution of computing resources.

This paper also gives an overview of OMWebBook and the potential integration points between these two web-based Modelica editors.

1.1 Terms and Acronyms used in this Paper

- Metamodel: defines the language and processes from which to form a model.
- DSML (Domain Specific Modeling Language): special-purpose languages designed to solve a particular range of problems.
- Strong relationship: child node is existence-dependent on the parent node.
- Meta-node: a node in a WebGME model hierarchy that is also part of the metamodel.
- Attribute: textual or numerical information as part of a node.
- Pointer: a named, directed, one-to-one relationship among nodes
- FCO: first class object that represents the atomic building block of a model in WebGME.
- ROOT: a container element that embeds all content of a project.

2 WebGME Background

WebGME is a web-based, visual modeling framework developed at Vanderbilt University (Maróti et al., 2014).The meta-programmable tool allows the creation of Domain Specific Modeling Language based modelers. It has a centralized, git-like model storage which allows multiple users to work on the same model simultaneously as well as creating separate branches that can be merged later. It also provides multiple extension points to allow fine-tuning of the user experience. The developer can define the visual appearance of different model elements or completely replace the entire model visualization. They can also implement their own model interpretation - like code generators or model verifiers - with the help of JavaScript based plug-ins.

3 Metamodeling

The WebGME framework provides a metamodeling paradigm resembling that of UML (Unified Modeling Language) (Lattmann et al., 2016). In contrast to traditional metamodeling frameworks, WebGME blurs the border between the metamodel and DSML models. Both are governed by the same datamodel that defines two strong relationships, *inheritance* and *containment*. These are single rooted trees over the same set of nodes with the inheritance root *FCO* (First Class Object) and the containment

DOI
10.3384/ECP18154220

root *ROOT*. The containment *ROOT* is not part of the inheritance tree nor the metamodel. The *FCO* is the prototypical *base* of all other nodes and defines the "name" *attribute*. Through prototypal inheritance, visualized as a red edge with a hollow arrow head in Figure 1, all nodes inherit the definition of the "name" attribute. Containment definitions are visualized as black edges with a filled diamond head. In Figure 1 the *FCO is* the *base* of *PortBase*, however a *ComponentBase can* contain a *PortBase*. All concepts in the metamodeling paradigm, except for inheritance, are definitions and should be view as *can have*s.

Pointer defintions in WebGME metamodeling environment are visualized as blue edges with an open arrow head as seen in Figure 2. *Connections* in WebGME are constructed using a pair of *pointers*, specifically *src* (source) and *dst* (destination). Nodes with these two pointer definitions are inferred to be *connections* and typically are visualized as edges when rendered in a DSML editor. The last metamodel concept used in this paper is the *Abstract* property. Nodes with this property cannot be instantiated in the model hierarchy. Even though a *Model* in Figure 1 can contain *ComponentBase*, the *ComponentBase* itself cannot be instantiated; however, the *Modelica.Mechanics.Translational.Components.Mass* from Figure 3 can be instantiated. For a full overview of the metamodeling concepts of WebGME refer to (Meijer and Mavridou, 2018) or (Maróti et al., 2014).

3.1 The Modelica Domain

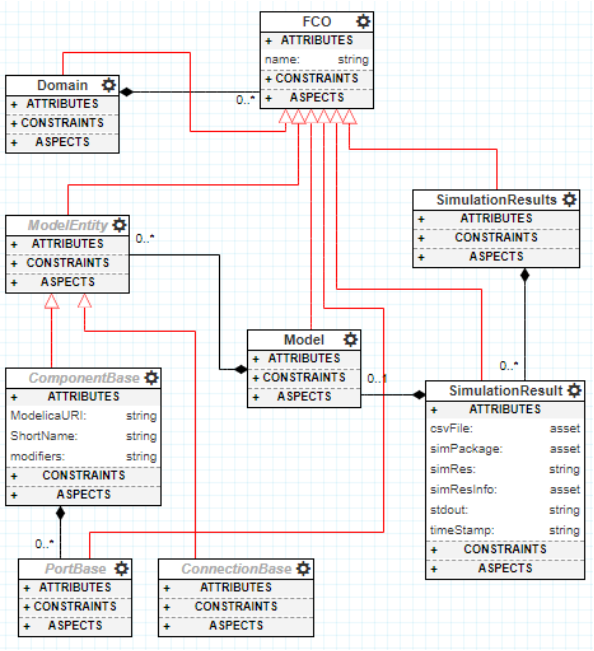

Figure 1. The base concepts for the Modelica Domain in WebGME

Modelica is at its core a declarative equation-based language extended with a rich type system, object-oriented features, functions, graphical annotations etc. (Fritzson, 2014). In close relation with the language is its standard library, the MSL (*Modelica Standard Library*). MSL is maintained by *The Modelica Association* and currently offers over 1300 components, many of which are ready-to-use, parameterized building blocks for modeling of complete dynamic systems. The target of the DSML and modeling environment presented in this paper is to offer a graphical editor building such systems. The approach taken here is to extract the interfaces and parameters from the MSL components and store this information as prototypes in WebGME. Before constructing any of these prototypes or meta-nodes, a metamodel with the base concepts was created. In Figure 1 the central concept is the *Model*, which can contain *ComponentBases* and *ConnectionBases*. In the finished editor the *Model* will act as the "drawing canvas" where dynamic systems of MSL components will be composed. *ModelicaURI* is an important attribute defined at the *ComponentBase*. In the derived MSL components in WebGME the *ModelicaURI* gets populated with the unique path to the associated MSL component.

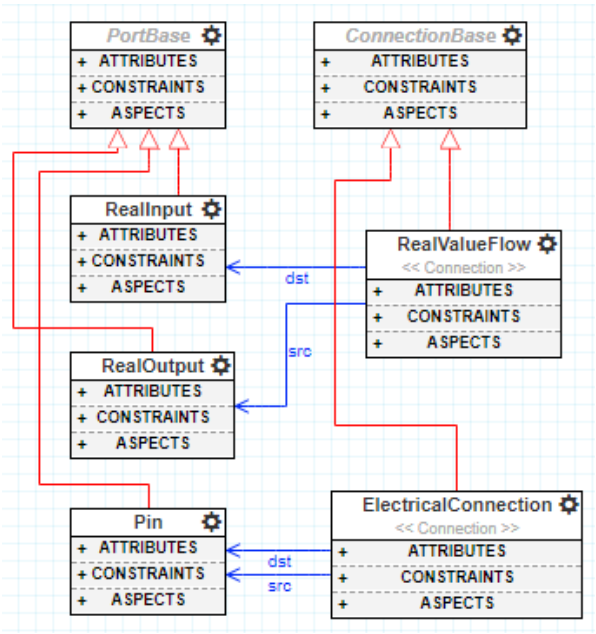

Figure 2. Metamodel for the Modelica connectors

On the left-hand-side of Figure 3 a sample of MSL components in WebGME are presented from the meta-view. To avoid name collisions, the *ModelicaURI* is also used as the name attribute of the meta-nodes. On the right-hand-side of Figure 3 the same components are displayed from the containment view. In contrast to the meta-view, this view shows what actual characteristics these prototypes *do* have, rather than what they *can* have. Those familiar with MSL probably recognize the flange connectors appearing as semicircles along the borders of the boxes. These connectors are not technically part of the metamodel. When prototypes are instantiated by the user on the canvas (mid section in Figure 5) the instances inherit all properties of their prototype. This not only includes the default values of the attributes, but also the

meta-definitions and the contained children nodes.

The MSL components are imported using a script that extracts the connectors and parameters from the Modelica code using the OpenModelica parser. WebGME nodes are created from the extracted data and organized in *Domain*-folders. The base metamodel contains predefined *Ports* and *Connections* that map to connectors of MSL for the purpose of capturing compatibility between different Modelica connectors. In Figure 2 a sample of these are presented. In the case of acausal connections, where direction does not matter, the source and destination are interchangeable. For directed value flows however, the connections are forced to be made from output to input. Constraints like these are captured by the WebGME API during modeling and although not necessary for a Modelica domain - a single port and connection concept would suffice - it allows for a more guided user-experience.

Up to this point the portion of the metamodel needed for governing the composition of systems of Modelica components from the Modelica Standard Library has been explained. The WebGME-DSS framework supports generation and simulation of Modelica code, see section 4.3, and in order to provide a closer link between models and simulation results the concept of *SimulationResult* is introduced, see Figure 1. Instances of these are created whenever a Model is invoked to be simulated. As the simulation progresses the different attributes are populated with the inputs and outputs to and from the Modelica compiler and runtime environment. In order to not overflow the models with large amount of data the *asset* attribute of WebGME is utilized. The asset attribute only stores a lightweight SHA-256 hash in the model acting as a key to the underlying artifact (typically stored at the file-system of the server). To maintain a mapping between the results and models, the *SimulationResult* node is populated with a copy of the Modelica model enabling the user-interface to map the plotted variables back to the graphical model.

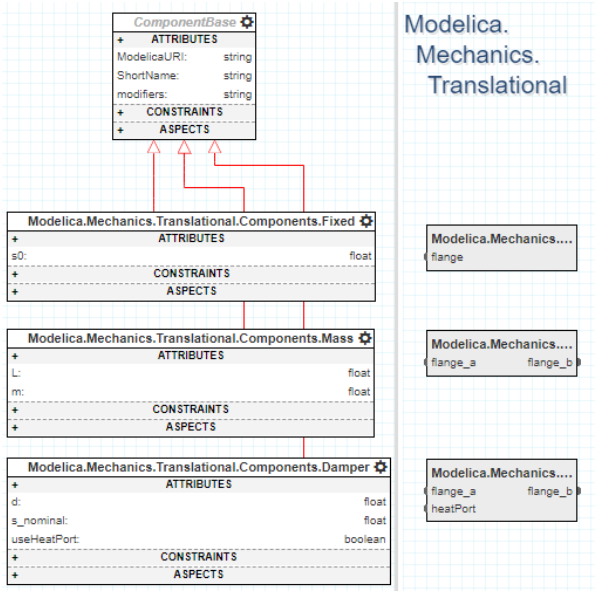

Figure 3. MSL models from the meta- and containment-perspective.

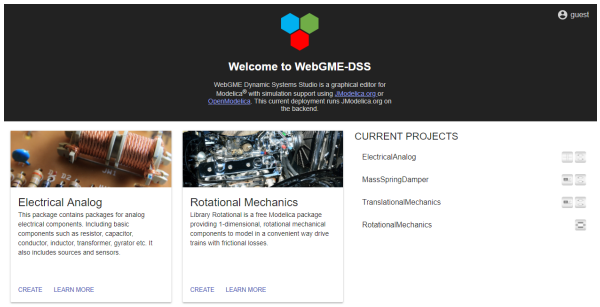

Figure 4. The landing page listing the user's current projects.

4 The User Interface

In the current section we will introduce the WebGME Dynamic Systems Studio, that has the goal of introducing Modelica on a beginner level by allowing visual composition of dynamic systems composed of components from the Modelica Standard Library.

4.1 Project Organization

The landing page - shown on Figure 4 - has two main functionalities. The user can create a new project either by selecting a specific domain from MSL as the basis of the project or by choosing the *Hybrid Domain* that allows selection of multiple libraries. These selections are not final, and only instructs the system to seed the project with the necessary information and can be changed any time during model editing.

The page also lists all the projects that the user has access to. The items not only show the name of the project, but highlight the applied libraries, giving extra information to the user. To modify access rights of projects, create organizations, view information about other users on the same deployment, etc. a link to the default WebGME profile page is available in the top-wright corner.

4.2 Modeling

The main page of the tool is the editor itself (Figure 5 and 6). It has two views - selectable on the bottom center of the page - the *Modeling* where the user can compose and initiate simulations of the system, and the *Results* where the progress of simulations and finally the time traces of the simulated variables can be plotted. The toolbar at the top provides zooming function to help in the navigation on larger, more complex systems. It also have a saving button to allow creating notes for the current version of the model.

The left sidebar hosts a part-browser where the elements of the used libraries are listed. These elements can be instantiated and put into the system by a simple drag and drop onto the main canvas. In addition to the available components, the sidebar also implements several buttons for important features. The first in the list is a check function, that verifies if a syntactically correct Modelica

DOI
10.3384/ECP18154220

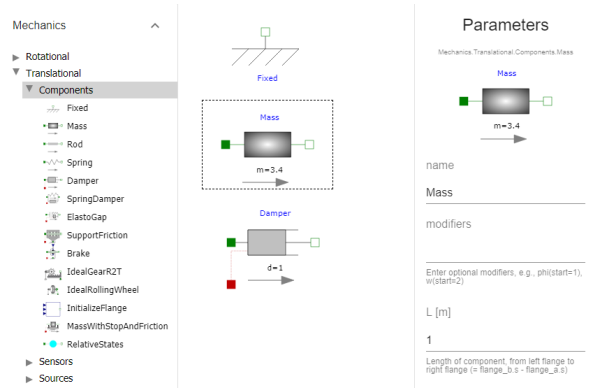

Figure 5. From left to right; the part-browser, canvas and attribute/parameter editor.

code can be generated from the model. If something is wrong - like two elements have the same name - the result dialog list the error providing a link, that will select the faulty node. The second button initiates a simulation. First, the user provides some basic parameters regarding the simulation, then, depending on how the server is configured, the collection of the generated necessary artifacts are either downloaded or the simulation gets started on the server. The third button brings up a multi-selection list of the available libraries. The final button shows the history dialog. The default view of the dialog lists the versions that were marked by the user - with the help of the save button - plus the current state of the project. As the WebGME creates micro-commits and stores even the smallest change, the user can switch to a *detailed* view, where all the created commits are visible. For every entry of the list, the user can reset the work to that given state or the difference between that and the current version can be visualized.

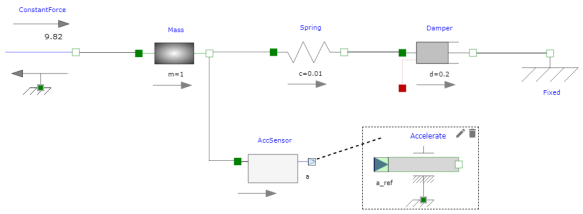

Figure 6. The port compatibility defined in the metamodel reflected in the modeling view.

With components on the main canvas, the user can edit the parameters by double-clicking on it or selecting and clicking the edit button. Hovering over the interfaces of the nodes, the user can initiate the creation of a connection. In connect mode, a dashed line from the source interface to the mouse pointer notifies the user about the ongoing task. Whenever the user reaches a valid target port-node, those interfaces will be highlighted guiding the user. If clicked on a correct endpoint, the connection is made while leaving the main canvas or clicking anywhere else cancels the operation. Each element visualizes its parameters according to its specification to give the user the most information upfront.

4.3 Simulation and Results

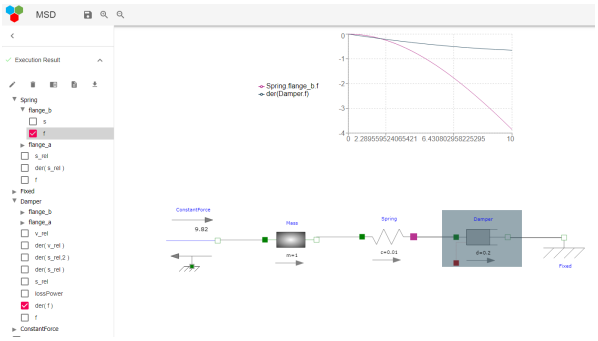

Figure 7. The plotted variables are highlighted in the originating model.

The *result* view of the tool provides the same areas, but with modified functionalities. The side-bar contains the list of the simulations. The items are highlighted according their state - whether they are running or completed. Once the simulation is finished, the structured list of variables becomes available, and by selecting the interesting ones, the user will initiate a plot visualization. The plot is visualized in the top half of the main canvas, while the bottom half shows the model. The model version shown is a copy of the one at the time of the start of the simulation - and this view is read-only - and the portions that own the plotted variables are highlighted with matching colors to inform the user about the content of the chart. To allow the user to analyze the result in depth, the chart can be detached from the screen - with the button at the top right corner - which changes the upper portion of the main canvas allowing the creation of multiple charts and individual control of their content. Once the user is done with the detached graphs they can be closed with a single button click. If the server run into issues during execution or the simulation is not ready, the top half of the main canvas will show the console output of the execution of the simulation engine.

5 System Architecture

There are a number of interacting components of the system that provides the user the WebGME-DSS interface described in the previous section. In the center of the system lies the WebGME server. Multiple instances can serve a single deployment which allows greater capacities in terms of connected users and a way of horizontal scaling. To enable multiple servers to work together, a Redis memory database needs to be added to the system. Through a publish/subscribe service, the Redis (Redis) provides fast and efficient communication between servers that allow notification to users who might be working on the same project but connected to different servers. Also an HTTPS reverse proxy - like Ngnix (NGINX) - becomes necessary

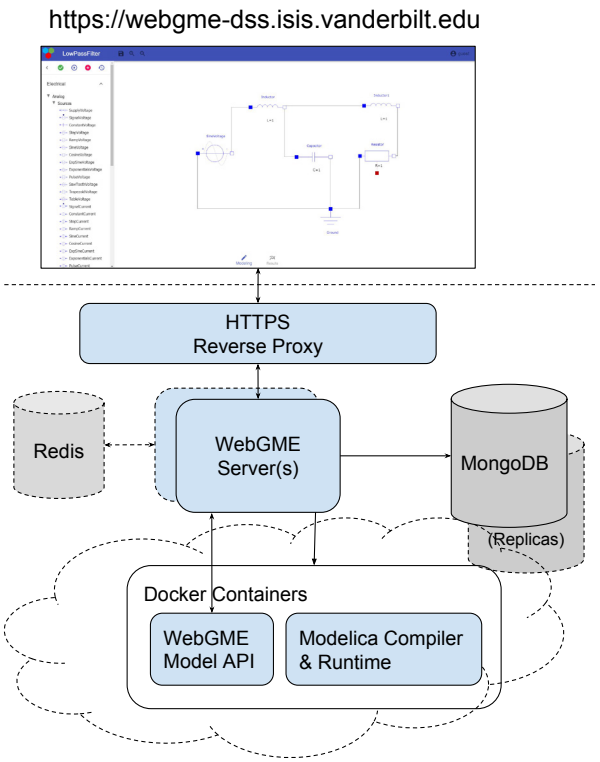

Figure 8. Overview of the different applications and services making up the framework.

to route the requests based on their session id, enabling users to communicate with a single server. This proxy also enhances the security of communication so its use should be considered in a single server configuration as well. MongoDB (MongoDB) is responsible for storing all project related data and multiple instances - so called shards - can be leveraged for bigger scalability. The final actors of the system are the Modelica simulators. The server connects to them and runs the simulation, gather the results. They are usually packaged in a Docker (Docker) container for easy deployment and minimal configuration. when it comes to vertical scaling, these components can live in a single computer or spread throughout multiple ones.

6 OMWebBook

There is currently a strong trend in integrating documents and computation. This is most visible in web technology where computational facilities are increasingly being embedded within web pages. However, facilities for user programming and mathematical modeling are still largely absent in web pages. Mathematica (Wolfram, 2003) pioneered the idea and implementation of active interactive electronic notebooks. The Mathematica notebook facility is an interactive WYSIWYG (What-You-See-Is-What-You-Get) realization of Literate Programming, a form of programming where programs are integrated with documentation in the same document. However, the original implementation of Literate Programming was a non-interactive system integrated with the document processing system LaTex.

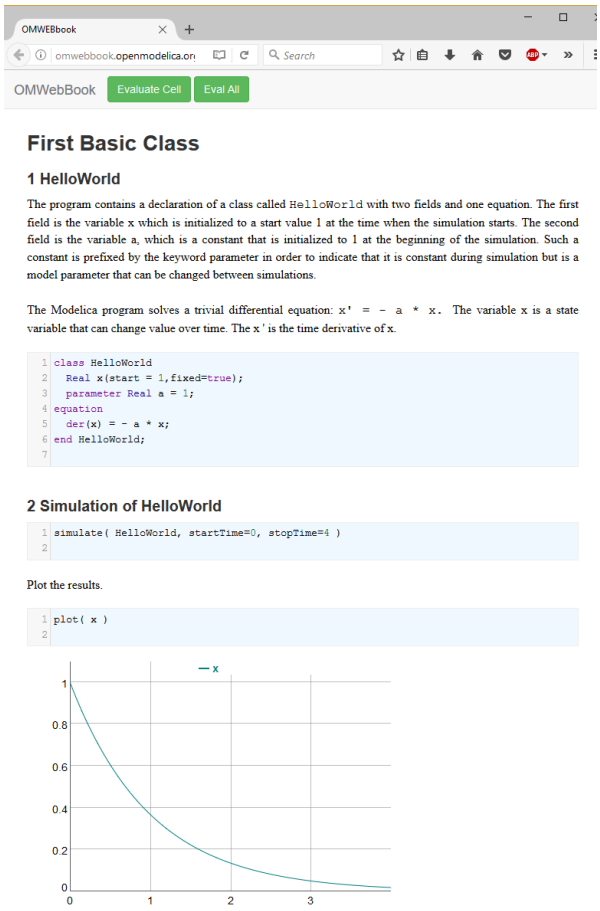

Figure 9. OMWebBook with editable models, simulations, and plots.

Traditional documents, e.g. books and reports, essentially always have a hierarchical structure. They are divided into sections, subsections, paragraphs, etc. Both the document itself and its sections usually have headings as labels for easier navigation. This kind of structure is also reflected in electronic notebooks. Every notebook corresponds to one document and contains a tree structure of cells. A cell can have different kinds of contents, and can even contain other cells. The notebook hierarchy of cells thus reflects the hierarchy of sections and subsections in a traditional document. The OMNotebook notebook facility in OpenModelica supports several kinds of contents, for example cells with executable Modelica model classes, commands, documentation, pictures, and 2D graphs. However, OMNotebook is an off-line tool, part of the OpenModelica tool suite, that has to be installed before use.

Therefore, we have developed OMWebBook (Fritzson, 2017), (Figure 9) which is an on-line interactive web-based electronic book (`http://omwebbook.openmodelica.org/`). This is similar to OMNotebook, but textual model editing and simulation is per-

DOI
10.3384/ECP18154220

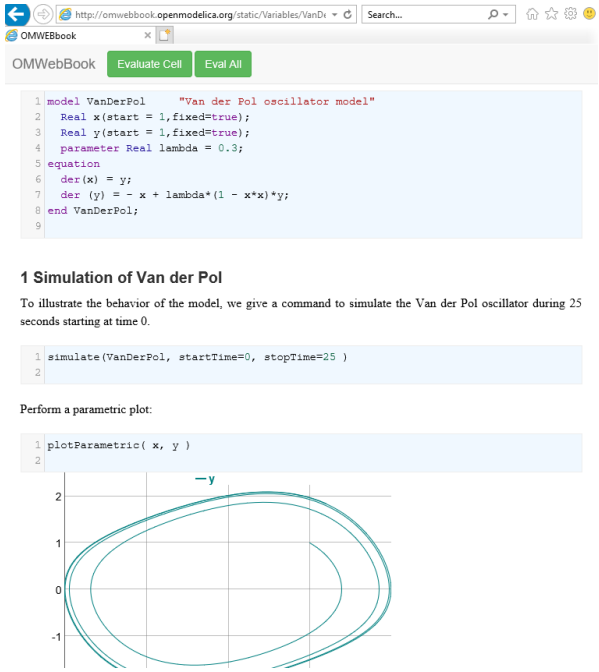

Figure 10. The VanDerPol sub-document showing a cell with a Modelica model, simulation commands, and plot results.

formed in a web-browser. Simulation is performed by a dedicated simulation server. Thus, the user need not install OpenModelica on a computer in order to edit models and run simulations. Editing and simulation can even be done from smartphones or tablets. Figure 9 shows part of the OMWebBook introductory Modelica teaching material called DrModelica (Lengquist-Sandelin et al.), starting with the simplest possible HelloWorld model that the student can edit, simulate and plot in the web browser.

Figure 10 shows the model, documentation, simulation and plotting of the VanderPol model.

However, so far OMWebBook has had one big drawback - it has lacked support for web-based graphic editing of Modelica models. This can be seen in the chapter about simplified modeling of electric and hybrid vehicles, starting with the simple electric vehicle graphical model depicted in Figure 11.

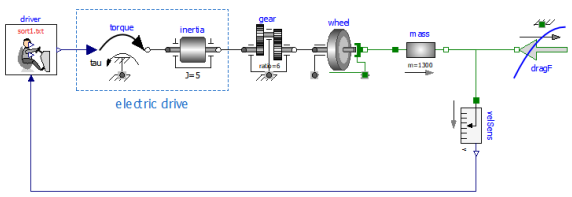

Figure 11. Connection diagram of a first, very simple, electric vehicle model.

The reader was previously instructed to download and install OpenModelica in order to edit and simulate this graphic model. However, this is no longer necessary with the combination of the WebGME web-based graphic model editor and the OMWebBook text editing and simulation capabilities. Now, also graphical models can be embedded in an on-line electronic book and edited and simulated on the web.

7 Future Work

This proof of concept implementation proved to be an efficient integration platform that can bring Modelica closer to end-users and allow them to focus solely on the system they try to describe. By integrating analysis tools we plan to provide a broader spectrum of functions to the users. Also, by better integrating the OMWebBook code-editor into the WebGME-DSS the user will be able to describe more detailed behavior for the components of their systems. The code editing also supports syntax highlighting as well as checks for compilation errors. Future work will address the enlargement of available components. This can be done in multiple ways. More MSL elements will be curated into the system and a library importer will allow developers to include their custom Modelica libraries. Finally, a special visualization will provide a platform for the user to create new components by using inheritance and a visual description language.

References

Docker. Docker - Build, Ship, and Run Any App, Anywhere. `https://www.docker.com/`. Cited 2018 May 11.

Peter Fritzson. *Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. Wiley IEEE Press, 2014. ISBN 9781-118-859124.

Peter Fritzson. Introduction to Modelica with Examples in Modeling, Technology, and Applications using OMWebBook `http://omwebbook.openmodelica.org/`. Technical report, Linköping University, PELAB - Programming Environment Laboratory, 2017.

Peter Fritzson and Vadim Engelson. Modelica — a unified object-oriented language for system modeling and simulation. In Eric Jul, editor, *ECOOP'98 — Object-Oriented Programming*, pages 67–90, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. ISBN 978-3-540-69064-1.

Zsolt Lattmann, Tamás Kecskés, Patrik Meijer, Gábor Karsai, Péter Völgyesi, and Ákos Lédeczi. Abstractions for modeling complex systems. In *International Symposium on Leveraging Applications of Formal Methods*, pages 68–79. Springer, 2016.

Eva-Lena Lengquist-Sandelin, Susanna Monemar, Peter Fritzson, and Peter Bunus. Drmodelica - a web-based teaching environment for modelica.

Miklós Maróti, Tamás Kecskés, Róbert Kereskényi, Brian Broll, Péter Völgyesi, László Jurácz, Tihamer Levendovszky, and Ákos Lédeczi. Next generation (meta) modeling: Web-and cloud-based collaborative tool infrastructure. *MPM@ MoDELS*, 1237:41–60, 2014.

Patrik Meijer and Anastasia Mavridou. How to build a design studio with webgme. 05/09/2018 2018. ISSN ISIS-18-101.

MongoDB. MongoDB: MongoDB for GIANT Ideas. https://www.mongodb.com/. Cited 2018 May 11.

NGINX. NGINX: High Performance Load Balancer, Web Server, Reverse Proxy. https://www.nginx.com/. Cited 2018 May 11.

Redis. Redis. https://redis.io/. Cited 2018 May 11.

Stephen Wolfram. *The Mathematica Book*. 5th Ed. Wolfram Media Inc, 2003. ISBN 1579550223.

DOI
10.3384/ECP18154220