OMWeb – Virtual Web-based Remote Laboratory for Modelica in Engineering Courses

Mohsen Torabzadeh-Tari, Zoheb Muhammed Hossain, Peter Fritzson, Thomas Richter[1]
PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden
{mohto, x10muhho, petfr }@ida.liu.se
[1]Rechenzentrum, Stuttgart University, Germany
[1]richter@rus.uni-stuttgart.de

Abstract

In this paper we present a web-based teaching environment, OMWeb, useful both in engineering courses as well as for teaching programming languages. OMWeb can be an alternative or complementary tool to the traditional teaching method with lecturing and reading textbooks.

Experience shows that using such interactive platforms will lead to more engagement from the students. With such a solution the student can focus more on the important learning goals. The student needs only to open a web browser and start writing programs in order to use the tool. The OMWeb server contains all the needed software. In each interaction, the server returns results to the user. This solution allows each student to work at his/her own speed, at any time, and remotely, enhancing the individual learning.

OMWeb is part of the open source platform OpenModelica. It can be applied to several areas in natural science, such as physics, chemistry, biology, biomechanics etc., where phenomena can be illustrated by dynamic simulations.

Keywords: OMWeb**,** *OpenModelica, Virtual, Web-based*

1 Introduction

In this paper we introduce a learning environment for web-based modern object-oriented equation-based modeling and simulation. This environment, called OMWeb, is useful both in programming language teaching and in engineering courses. The primary application shown in this paper is teaching the Modelica language [1]. However, the concept can also be adapted to other languages. In this way the student has an interactive common platform for learning programming languages as well as learning through virtual experiments with physical phenomena.

This kind of interactive course allows experimentation and dynamic simulation as well as execution of computer programs. As a part of the open source platform OpenModelica, [2], this makes it possible to integrate applied sciences in physics, human biology [3], mathematics, and computer science.

2 OpenModelica Platform

In 2002 an initiative was taken by the PELAB group at Linköping University to develop an open source platform for the Modelica language, to be called OpenModelica [2],[3] and [10]. The OpenModelica effort has expanded, and is in recent years also supported by the Open Source Modelica Consortium.

The OpenModelica environment, shown in Fig. 1, consists of several interconnected subsystems. The debugger currently supports debugging of an extended algorithmic subset of Modelica, MetaModelica.

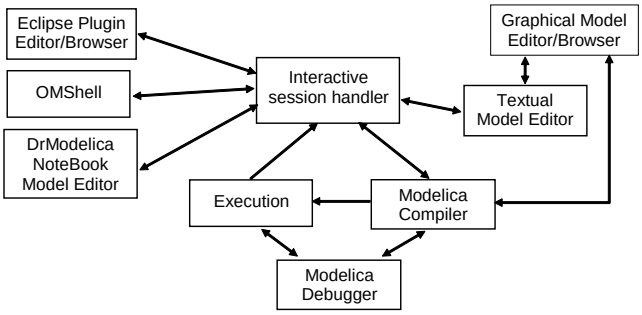

Figure 1. Illustration of communication between different parts of the OpenModelica platform.

The OpenModelica Notebook editor, OMNotebook (see Section 5.3), provides an active electronic notebook including an editor. The notebook is active in the sense that models inside the book can be changed and executed, it is not just a passive textbook or html page. This is one of the first open source efforts that makes it possible to create interactive books for educational purposes in general, and more specifically for teaching and

learning programming. Traditional teaching methods with lecturing and reading a textbook are often too passive and don't engage the student as much.

3 OMWeb architecture

The OMWeb architecture is composed of three decoupled set of entities namely the *Teacher* and/or *Student Client* (TC and SC), the *E-learning Community Server* and the *Computation Client*, similar to the NumLab architecture (Section 5.1).

The three layers have been developed in different programming languages; the clients are developed in Java, the E-learning Community Server (*ECS*) is developed in Ruby On Rails and for the Computation Client (CC) C++ is used, see Fig 2.

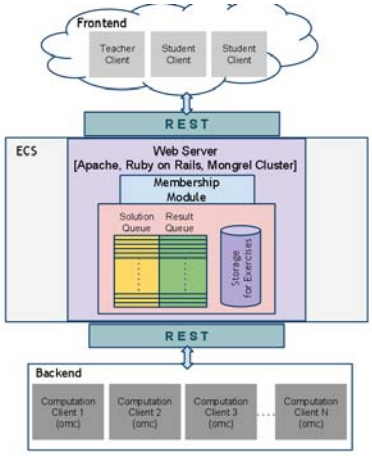

Figure 2. OMWeb architecture

Communication among the modules is done following the REpresentational State Transfer (REST) architecture; which uses Hyper Text Transfer Protocol, HTTP as the carrier of messages across the network. The HTTP has four methods for accessing and updating the resources - GET, POST, PUT and DELETE.

Moreover, JavaScript Object Notation (JSON) is used to format the data for communication. One reson behind the choice of JSON format is that it is easy for the humans to read and write as well as for the machines to parse the data.

4 OMWEb – OpenModelica Virtual Web-based Learning Platform

In this Section the different parts of OMWeb are explained in detail.

OMWeb provides a programming environment within a web browser. This facilitates for the student to learn and participate in courses, independent of time and place. The availability in a standard web browser makes it easier to get started with than if you have to install special software packages.

The *Student Client* and the *Computation Client* are both active entities in the system, whereas ECS is the passive entity. In detail, the ECS never initiates a communication, rather it only responds to occurring events.

On the other hand the end clients are the ones who are always polling for Solution or Result messages from the Solution and Result Queue of the ECS as soon as they finish POST-ing a message to the ECS. The communication diagram in Fig 6 reflects a better illustration of the message flow in the System.

4.1 Frontend: Teacher Client, TC

The TC [13] is the web frontend of the system and is specially developed for a teacher to post his/hers exercises or assignments intended for the students to solve and get evaluated.

Fig. 3 illustrates the Graphical User Interface of this client. The tab "Exercise Generator " has three fields for entry; the name of the exercise, the description of the exercise and the text area takes in the program code.

The teacher then clicks on the "Send Exercise" button; this action will first generate the JSON script of the corresponding exercise by filling in the "value" tag of the respective "identifier". Later, it sends the JSON string using the HTTP POST which posts the exercise to the ECS.

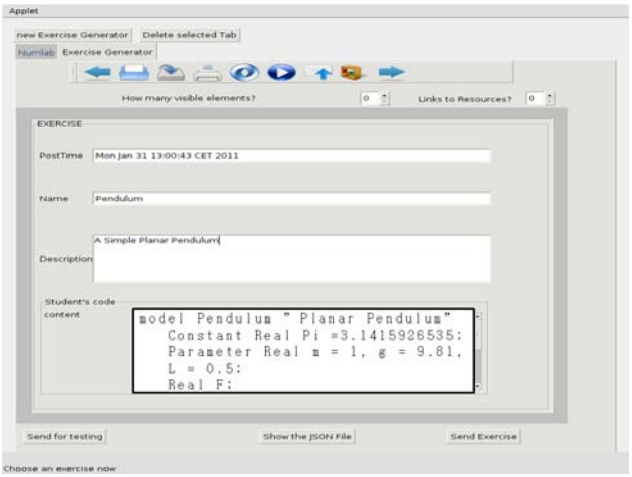

Figure 3. Teacher Client

4.2 Frontend: Student Client, SC

The SC [14] is a web frontend designed for the students who are to solve the exercises posted by the TC, see Fig 4.

A student first opens the web applet and clicks on the drop down box of the client user interface that is

labeled "Fetch Exercise" to retrieve the list of exercises posted by their teacher. The student then selects one of the exercises from the list and the exercise is shown in the client interface.

The exercise may contain one or more sections labeled as Editable and Non-editable where the students can edit the code of the Editable section(s) only.

Later, when the student is done with solving the program, the button labeled "Execute" is pressed, this will first generate the JSON string for the Solution of the exercise and then will send over the network to the ECS using the HTTP POST method.

As soon as sending of the solution is done the SC initiates to poll for the Result of the Solution from the ECS.

The client continues to poll until it receives the Result JSON string from the ECS. On receiving the Result the JSON string is parsed and the result data is shown on the output section of the client interface.

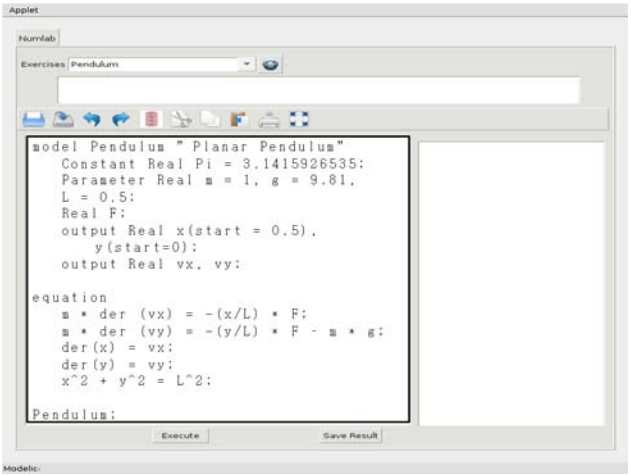

Figure 4. Student Client

4.3 Middleware, E-learning Community Server, ECS

The ECS module works as the middleware between the frontend client and the backend client (server) in order to forward the requests back and forth.

The ECS is composed of three internal modules which are used to manage the messages of the clients: Community, Membership and Resources.

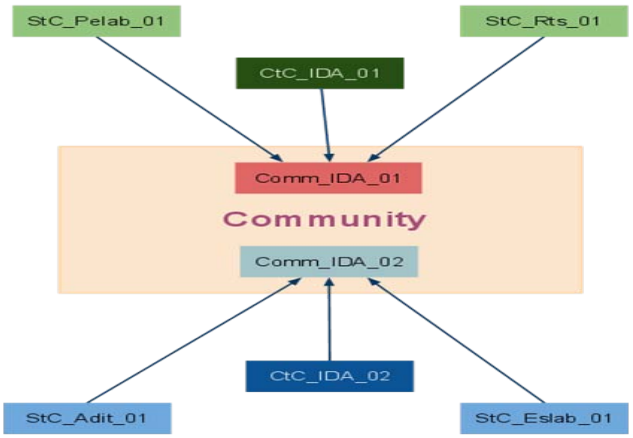

Figure 5. ECS Community

In order for the clients at both ends to communicate with each other they are required to be first registered to the ECS with a unique Membership ID where the membership is granted by the administrator of the ECS.

Next, the Members are assigned to a common Community (Fig. 5), this is required to route the messages of the same community members back and forth.

For example let us assume that there are four frontend SCs and two backend Computation Clients with membership IDs StC_Pelab_01, StC_Rts_01, StC_Adit_01, StC_Eslab_01 and CtC_IDA_01, CtC_IDA02 respectively. Also assume that there are two communities in the ECS Comm_IDA_01 and Comm_IDA_02.

Let us also assume that client StC_Pelab_01, StC_Rts_01 and CtC_IDA_01 are members of the Comm_IDA_01 community and that clients StC_Adit_01, StC_Eslab_01 and CtC_IDA_02 are members of the Comm_IDA_02 community.

Now, the messages sent by the clients StC_Pelab_01 and StC_Rts_01 can be processed by the backend client CtC_IDA_01 only and messages sent by the StC_Adit_01, StC_Eslab_01 would be processed by the CtC_IDA_02 only.

This kind of routing guarantees that the messages are received by the intended entities only and hence eliminates any misrouting possibilities.

The ECS also maintains a database to store the exercises posted by the teachers using an authentic TC. The exercises are given an unique ID in order to be identified by the font/backend clients while generating the Solutions and the Results, further about this is discussed in later segment.

To manage the messages from and to the clients two separate queues are maintained. These queues are event driven queues supplied by the Ruby On Rails architecture. An event handler takes care of the specific events generated by the incoming messages at the ECS. When

the ECS receives an incoming message from the Student Client containing the HTTP method POST, the event handler routes the message to the Solution Queue and when the ECS receives an incoming message from the Computation Client with the HTTP method POST, it is put to the Result Queue.

Similarly, when the incoming message from the Student Client and Computation Client is a HTTP GET, the messages are retrieved from the Result and Solution Queue respectively and forwarded to the clients who initiated the GET request.

4.4 Backend: Computation Client, CC

The CC is responsible for executing the Solutions sent by the SC, evaluate the correctness of the program code and send the Result back to the ECS.

As we have mentioned in the previous section, the clients who are member of the same community can communicate between themselves; so, the computation client CtC01 could only fetch, execute and evaluate the result that was sent by the Student Client StC01.

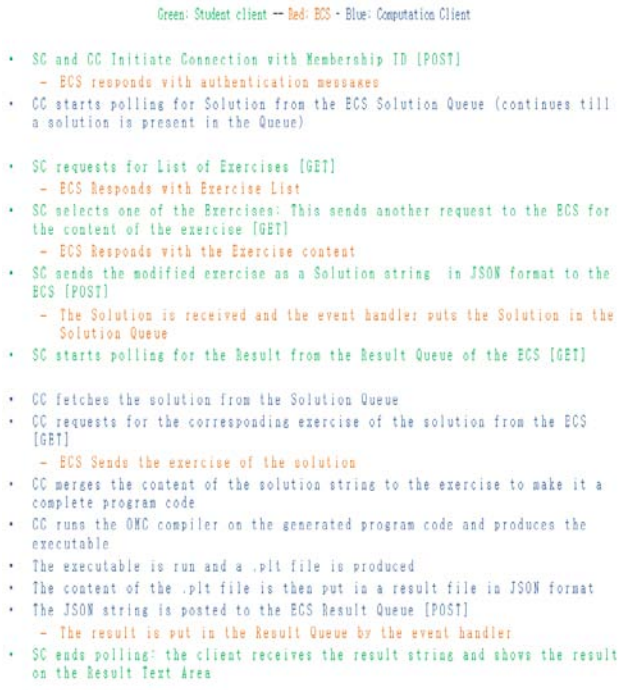

Figure 6. Sequence of message flow

The CC is developed in C++ on a Linux platform. It adopts Sandbox technique to limit the program instance accesses, e.g. only the Linux commands available inside the Sandbox. When the CC fetches a Solution message from the Solution Queue of the ECS it carries out several sequential tasks in order to evaluate the program code. First, it parses the solution JSON string and extracts the respective exercise of the solution from the ECS. Then the core solution content is extracted from the JSON string and is merged to the "editable" section of the exercise; which makes it a complete program. A file with the program's name is then created and the generated program code is copied and pasted into it.

Next, the type of compiler that should be used to compile the program is extracted from the JSON string with the respective flags and the program file is compiled, in our case it is the Modelica compiler. If there is any error during the compilation, a result JSON string is generated with the compiler error message and is posted to the ECS Result Queue.

Otherwise, on successful compilation a Makefile is produced which is then executed. The execution of the Makefile creates an executable, it is then executed and on successful execution, a .plt file is created.

The content of the .plt file is then pasted in the result JSON string and posted back to the ECS Result Queue, eventually which is polled by the specific Student Client.

5 Related Work

A brief survey is presented in this section covering some existing web-based and interactive learning platforms.

5.1 NumLab Architecture

At University of Stuttgart a web-based virtual environment NumLab [7] is available for computerized mathematical calculations including related subjects. The idea is to provide a web-based teaching environment where the students can focus on the numerical and mathematical topics without having to install any software packages.

NumLab is built according to the client-server architecture with a Java-applet in the front-end, a middleware layer E-learning Community Server, and a back-end client containing all the involved software packages, [9].

For communication between the ECS server and the other parts of the system REST, Representational State Transfer, is used which simplifies the communication in web-based distributed systems. The data representation and exchange format JSON, JavaScript Object Notation is used, [8].

5.2 Intelligent Tutoring System

There are many web-based intelligent tutoring systems that are worth to mention. For example the ELM-ART used for teaching the Lisp language [12] or a plugable web-based tutoring system in [11].

5.3 DrModelica

The OMNotebook subsystem in OpenModelica is currently being used for course material (DrModelica) in teaching the Modelica language and equation-based object-oriented modeling and simulation, (see Fig 7).

It can easily be adapted for use with electronic books teaching other programming languages. OMNotebook can also easily be used in other areas such as physics, biology chemistry, biomechanics etc., where phenomena can be illustrated by dynamic simulations within the book.

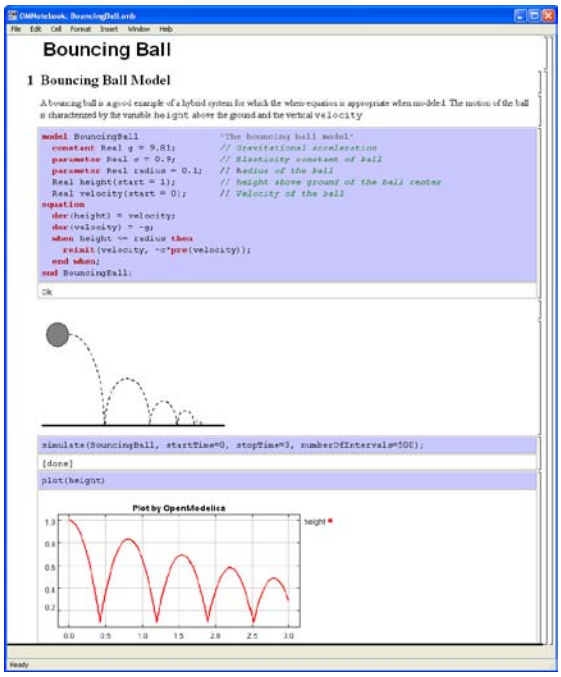

Figure 7. Bouncing ball example with movement animation in OMNotebook

5.4 OMScheme

With OMScheme the OMNotebook paradigm is generalized towards other programming languages than Modelica, e.g the Scheme programming language, [6]. An implementation of the factorial function using OMScheme is shown in Fig 8.

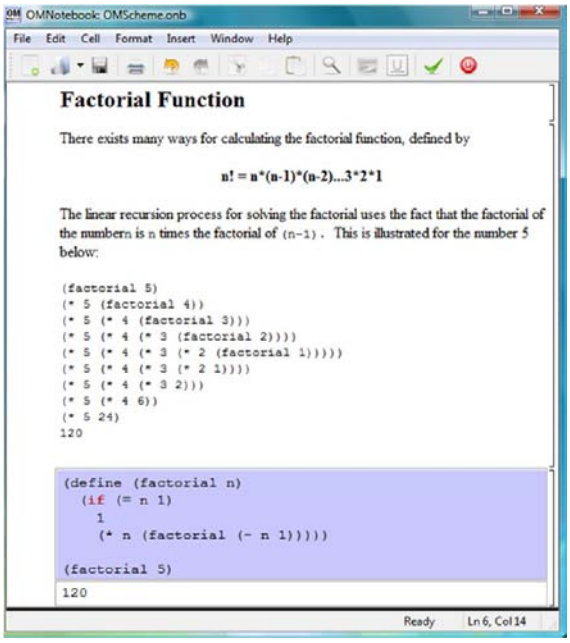

Figure 8. `Factorial function` illustrated in OMScheme

5.5 DrControl

DrControl, Fig 9, is a recently developed active electronic book course material based on OMNotebook for teaching control theory and modeling with Modelica.

It contains explanations about basic concepts of control theory along with Modelica exercises. Observer models, Kalman filters, and linearization of non-linear problems are some of the topics in the course used in control of a pendulum, a DC motor, and a tank system model among others.

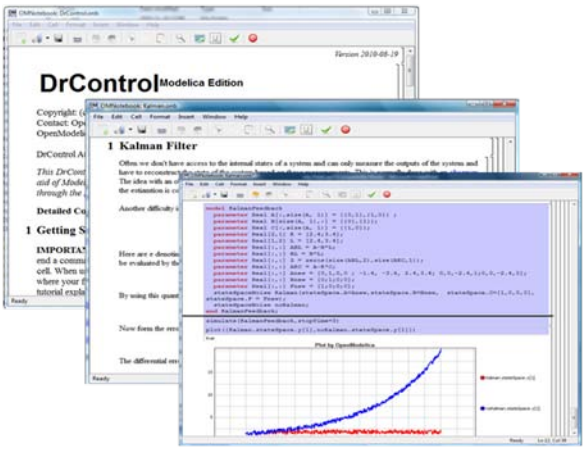

Figure 9. DrControl for teaching control theory concepts.

6 Future Work

The OMWeb platform presented in this paper is pure-text based. Integrating the graphical connection editor, OMEdit into OMWeb would be one of the desired next mile-stone. Also a 3D visualization and syntax highlighting should be supported for making the environment more user friendly.

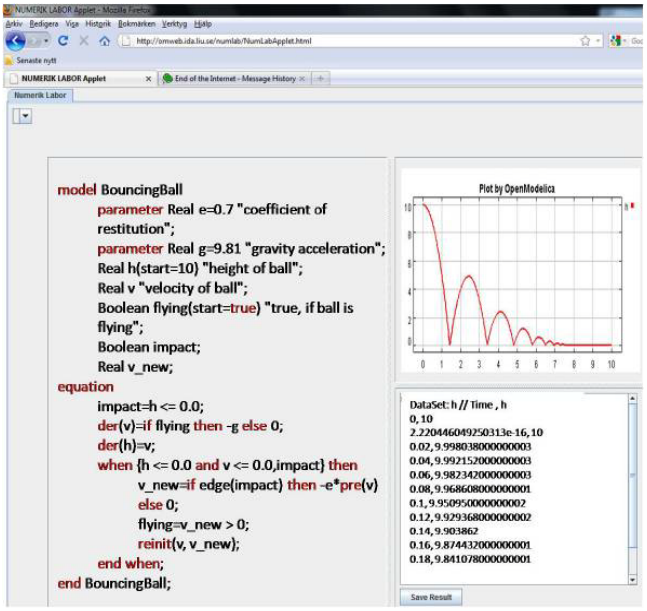

Figure 10. Bouncing Ball illustrated in OMWeb with syntax highlighting aid and plotting.

7 Conclusions

In this work we extended the basic idea of an active web-based teaching environment for educational purposes to handle multiple programming languages.

An early prototype is being developed for handling the Scheme and Modelica languages. OMWeb takes the virtual remote learning environment idea further by introducing OpenModelica platform within NumLab and widens the applicability to a wide range of engineering courses by introducing Modelica language in those courses.

The benefits and opportunities offered by a web-based solution is natural to access a vast amount of knowledge and information but also the ability for the student to work at his or hers own speed which enhances the learning process. Furthermore, the student activity is encouraged more by the interactivity and ease-of-use within an easy-to-use web-based interface

8 Acknowledgements

This work has been supported by EU project Lila and Vinnova in the ITEA2 OPENPROD project. The Open Source Modelica Consortium supports the OpenModelica work.

References

[1] Modelica Association. The Modelica Language Specification Version 3.1, May 2009. www.modelica.org

[2] Peter Fritzson et al OpenModelica Users Guide and OpenModelica System Documentation, www.ida.liu.se/projects/OpenModelica, 2009.

[3] Anders Sandholm, Peter Fritzson, Varun Arora, Scott Delp, Göran Petersson, and Jessica Rose. The Gait E-Book - Development of Effective Participatory Learning using Simulation and Active Electronic Books. In *Proceedings of the 11th Mediterranean Conference on Medical and Biological Engineering and Computing (Medicon'2007)*, Ljubljana, Slovenia, June 26 - 30, 2007.

[4] Bernhard Bachmann, Peter Aronsson, and Peter Fritzson. "Robust Initialization of Differential Algebraic Equations" In Proc. of (Modelica '06), Vienna, Austria, 2006.

[5] Mohsen Torabzadeh-Tari, Peter Fritzson, Adrian Pop, and Martin Sjölund. Generalization of an Active Electronic Notebook for Learning Multiple Programming Languages, IEEE EDUCON Education Engineering 2010 – The Future of Global Learning Engineering Education, Madrid, Spain, 2010

[6] Anders Fernström, Ingemar Axelsson, Peter Fritzson, Anders Sandholm, Adrian Pop. OMNotebook – Interactive WYSIWYG Book Software for Teaching Programming. In *Proc. of the Workshop on Developing Computer Science Education – How Can It Be Done*?. Linköping University, Dept. Computer & Inf. Science, Linköping, Sweden, March 10, 2006

[7] Bankolé Adjibadji, Stephan Rudolf , and Thomas Richter. Numerische Mathematik im Browser: Das Virtuelle Programmierlabor ViP, Stuttgart University, Rechenzentrum, Feb 2010 http://isblab.rus.uni-stuttgart.de:7070/numlab/exercises

[8] Douglas Crockford. Introducing JSON, URL: http://json.org. Retrieved March 25, 2010

[9] Heiko Bernloehr. E-learning Community Server, URL: http://freeit.de/ecsa/index.html, Retrieved March 25, 2010

[10] Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, and

David Broman. The OpenModelica Modeling, Simulation, and Software Development Environment. In Simulation News Europe, 44/45, December 2005. See also: http://www.openmodelica.org.

[11] Ang Yang, Kinshuk, Ashok Patel, A Plug-able Web-based Intelligent Tutoring System, Conference on Information Systems ECIS 2002, Gdansk, Poland

[12] Gerhard Weber, Peter Brusilovsky, ELM-ART: An Adaptive Versatile System for Web-based Instruction, International Journal of Artificial Intelligence in Education, 2001, Vol 12, pp 351-384

[13] http://omweb.ida.liu.se/TeacherClient

[14] http://omweb.ida.liu.se/StudentClient