DrModelica
A Web-Based Teaching Environment for Modelica

Eva-Lena Lengquist Sandelin, Susanna Monemar, Peter Fritzson, Peter Bunus
PELAB, Programming Environment Laboratory
Department of Computer and Information Science
Linköping University, S-581 83 Linköping, Sweden
Email: {evale, x02susmo, petfr, petbu}@ida.liu.se

Abstract

This paper states the need for interactive teaching materials for programming languages within the area of modeling and simulation. We propose an interactive teaching material for the modeling language Modelica inspired by existing tutoring systems for Java and Scheme.

The purpose of this new teaching material, called DrModelica, is to facilitate the learning of Modelica in a modeling and simulation environment. We have developed two versions of DrModelica, one that is based on Mathematica and another that is intended for the web. With the web version of DrModelica we hope for an increased usage of Modelica.

1. Background

The concept of model, system and experiment are central in the area of modeling and simulation. "A model of a system is anything an "experiment" can be applied to in order to answer questions about that system." [1] "A simulation is an experiment performed on a model." [1]

Tools that are used for modeling and simulation are becoming a powerful aid in the product development process. Using advanced tools and languages to build a model of the product and then simulate its behavior before producing a physical prototype, minimizes the number of errors that can occur during fabrication. This minimization consequently leads to a decrease in the time needed to develop the final product. Furthermore, the earlier the errors are detected, the cheaper the corrections are.

Not too long ago in the history of modeling and simulation technology, mathematical models were implemented by hand. The models were usually designed on paper using mathematical notation and the programs written manually in a high-level programming language, like C or Fortran, and stored in text files. Much manual work was needed, making not only maintenance of models expensive, but also the modification of models in order to adapt to new requirements [2].

2. Modelica

Modelica is a new language for hierarchical object-oriented physical modeling which is being developed through an international effort (Elmqvist et al. 1999 [3], Fritzson and Bunus 2002 [4], Fritzson 2003 [1]). The language unifies and generalizes previous object-oriented modeling languages. Modelica is intended to become a *de facto* standard.

A Modelica program is built from classes like in any other traditional object-oriented language. The main difference compared to traditional object-oriented languages is that instead of functions (methods) equations are used to specify the behavior. A class declaration contains a list of variable declarations and a list of equations preceded by the keyword `equation`. We illustrate below a class corresponding to a resistor (`Resistor`) and an alternative voltage source (`VsourceAC`) modeled in Modelica.

```
model Resistor
   extends TwoPin;
   parameter Real R;
equation
   R*i=v;
end Resistor;

model VsourceAC
   extends TwoPin;
   parameter Real VA=220;
    parameter Real f=50;
   protected constant Real PI=3.14;
equation
   v=VA*[1];
end VsourceAC;
```

It should be also noted that both classes are specializations through inheritance of the class called `TwoPin`. The natural inheritance in the Modelica language works by extending classes with new equations and variables. The `TwoPin` class that defines electrical components that have two pins is defined as follows:

```
model TwoPin
   Pin p,n;
   Real v,i;
equation
   v = p.v - n.v; 0 = p.i + n.i; i = p.i;
end TwoPin;
```

This class instantiates the class `Pin` twice. This is a special kind of class called connector class. Such connectors declare variables that are part of the communication interface of an object defined by the connectors of that object. Thus, connectors specify the interface for interaction between a component and its surroundings. Our `Pin` connector class uses two Real variables: one for the current (`i`) and one for the voltage (`v`). Since in an electrical circuit the current should always be summed when connecting two components, according to Kirchoff's law, the variable `i` defined in the `Pin` class will have the prefix `flow`.

```
connector Pin
   Real v;
   flow Real i;
end Pin;
```

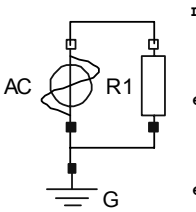
Besides the instantiation of two Pin interface objects, also called ports or connectors, some equations are provided that defines the behavior of the objects: such as the voltage drop along the component (`v = p.v - n.v`) or the current inside the component (`0 = p.i + n.i; i = p.i`). In a similar way a Ground component would be defined as follows:

```
model Ground
   Pin p;
equation
   p.v = 0;
end Ground;
```

Connections between objects can be established between connectors of equivalent type. Modelica supports equation-based acausal connections, which means that connections are defined as special equation forms. A connection equation form such as `connect(pin1,pin2)` with `pin1` and `pin2` of connector class `Pin`, connects the two pins so that they form one node. This is equivalent to, and is eventually expanded into two equations, namely:
`pin1.v = pin2.v; pin1.i + pin2.i = 0;`

The first equation states that the voltages of the connected wire ends are the same. The second equation corresponds to Kirchhoff's current law stating that the currents sum to zero at a node (assuming positive value while flowing into the component). The sum-to-zero equations are generated when the prefix flow is used. Similar laws apply to flows in piping networks and to forces and torques in mechanical systems.

Now we have all the components that are necessary to define the following model of a simple electrical circuit consisting of a sinusoidal voltage source and a resistor connected together.




```
model Circuit
   Resistor R1(R=10);
   VsourceAC AC;
   Ground G;
equation
   connect(AC.p,R1.p);
   connect(R1.n,AC.n);
   connect(AC.n,G.p);
end Circuit
```

Figure 1. Simple electrical circuit model.

3. Interactive Environments

In order for the Modelica language to be used to solve problems, a modeling and simulation environment is needed. In this section the environment MathModelica is introduced. MathModelica is integrated with Mathematica, a system which is described below.

3.1. Mathematica

Mathematica [5] is a computer algebra system and programming environment for performing mathematical computations. The system can be used in many different ways; the most basic functionality is to use it as a "calculator". The user types a calculation and Mathematica performs it immediately. However, there is a large difference between what a traditional calculator can do and what Mathematica can perform. Mathematica seamlessly integrates a numeric and symbolic computational engine, graphics system, programming language, documentation system, and advanced connectivity to other applications.

Mathematica can also be used as a modeling and simulation environment. When a model is simulated

in the environment, the results can be visualized in various ways, e.g. using the `Plot` function.

Mathematica is divided into two distinct parts: the computer algebra engine and expression interpreter ("kernel") that receives and evaluates all expressions sent to it and the user interface ("front-end"). The front-end provides the interface to the user and is concerned with such issues as how input is entered and how computation results are displayed to the user.

Mathematica's front-end documents are called notebooks. A notebook can contain specific computations, text (including hyperlinks to other Notebooks), graphics, sounds and animations. Using a hierarchical structure divided into sections, subsections etc., a notebook can be made to look like a traditional typeset document, with the advantage that the calculations can remain active and can be re-evaluated at any time.

3.2. MathModelica

MathModelica, from MathCore Enginerring AB [6], is a powerful engineering environment for physical modeling, simulation, analysis and design [7, 8]. In MathModelica, models are described using the Modelica language. Dymola [9], developed by Dynasim [10], is another powerful Modelica environment.

The MathModelica environment integrates Modelica-based modeling and simulation with graphic design, advanced scripting facilities, integration of code and documentation, and symbolic formula manipulation provided via Mathematica. Import and export of Modelica code between internal structured and external textual representation is supported by MathModelica. The environment extensively supports the principle of literate programming and integrates most activities needed in simulation design: modeling, documentation, symbolic processing, transformation and formula manipulation, input and output data visualization.

The user-interface of MathModelica consists of three main parts, the Model Editor, the Simulation Center and Mathematica notebooks. The Model Editor is a graphical tool to design models using predefined library components, see Figure 2. The current version of the editor is an extension of Microsoft Visio, which is a tool for technical diagramming and drawing. The Simulation Center is a graphical user interface for running simulations and plotting the simulation results of the models [7, 8, 11]. Mathematica notebooks are used in MathModelica to provide a text based programming environment.

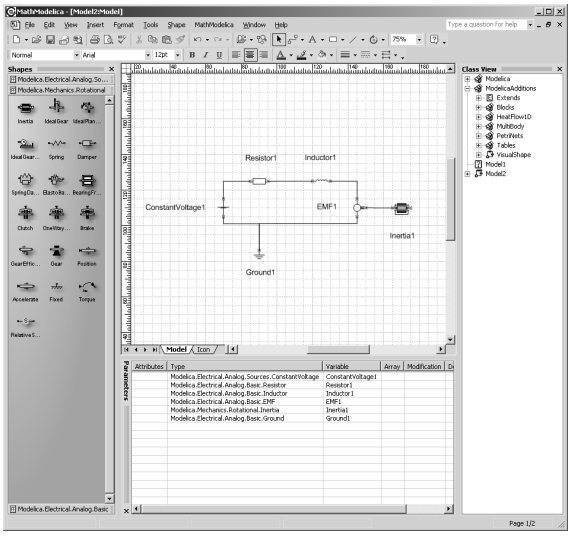


Figure 2. The Model Editor.

4. DrModelica

Understanding programs is hard, especially code written by someone else. For educational purposes it is essential to be able to show the source code and to give an explanation of it at the same time. Moreover, it is important to show the result of the source code's execution [12]. In modeling and simulation it is important to have the source code, the documentation about the source code, the execution result of the simulation model, and the documentation of the simulation results in the same document. The reason is that the problem solving process in computational simulation is an iterative process that often requires a modification of the original mathematical model and its software implementation after the interpretation and validation of the computed results corresponding to an initial model.

Most of the environments associated with equation-based modeling languages focus more on providing efficient numerical algorithms rather than giving attention to the aspects that should facilitate the learning and teaching of the language. There is a need for an environment facilitating the learning and understanding of Modelica. Users are reluctant to using a programming language that does not provide an adequate programming environment [13].

We have developed DrModelica [14] based on the MathModelica environment [6] and the ideas of Literate programming [15]. Literate programming is a programming methodology that was introduced by Donald E. Knuth. It represents the idea of organizing a source program in an "essay" manner by combining the source code with the corresponding documentation in the same document. By doing so it is easier to read and understand the program.

MathModelica has an interface allowing the user to write source code as well as documentation in the same document. The user does not have to switch to a command prompt to compile the source code, since this can also be performed in the MathModelica environment. The same document also contains plots of the simulation results. Additionally, in DrModelica the whole Modelica language is available to the user, unlike many other tutoring systems, where it is common to only provide a subset of the language.

DrModelica is a hierarchical structure of Mathematica notebooks. The most important notebook is similar to a table of contents that holds all other notebooks together by providing links to them. This particular notebook it the first page the user will see (Figure 3).

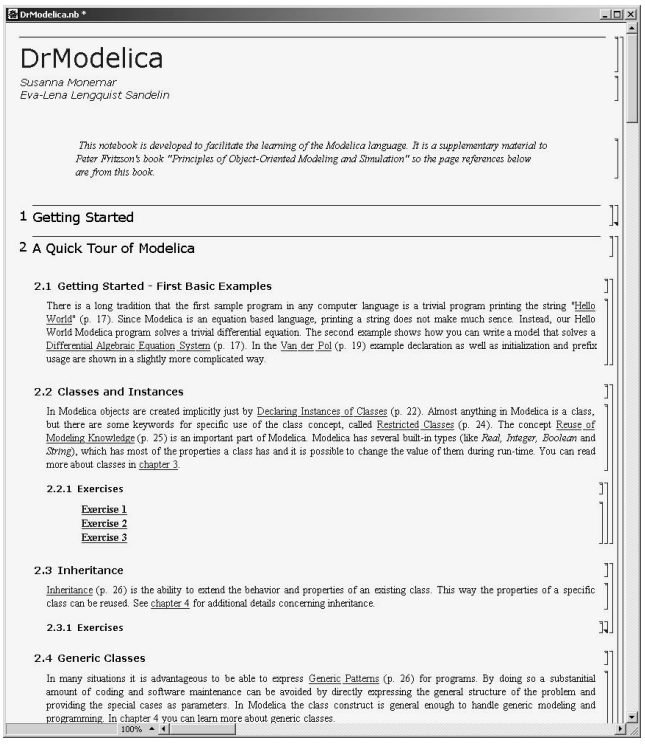


Figure 3. The main page of DrModelica.

In each chapter of DrModelica the reader is presented with a short summary of the corresponding chapter in the book "Principles of Object-Oriented Modeling and Simulation with Modelica" by Peter Fritzson [1]. The summary introduces some *keywords, being hyperlinks* that will lead the reader to another notebook describing the keyword in detail.

Now, let us consider that the link "HelloWorld" in section 2.1 in Figure 3 is clicked. The new notebook, to which the user is being linked (see Figure 4), is not only a textual description but also contains one or more examples explaining the specific keyword. In the class HelloWorld a differential equation is described.

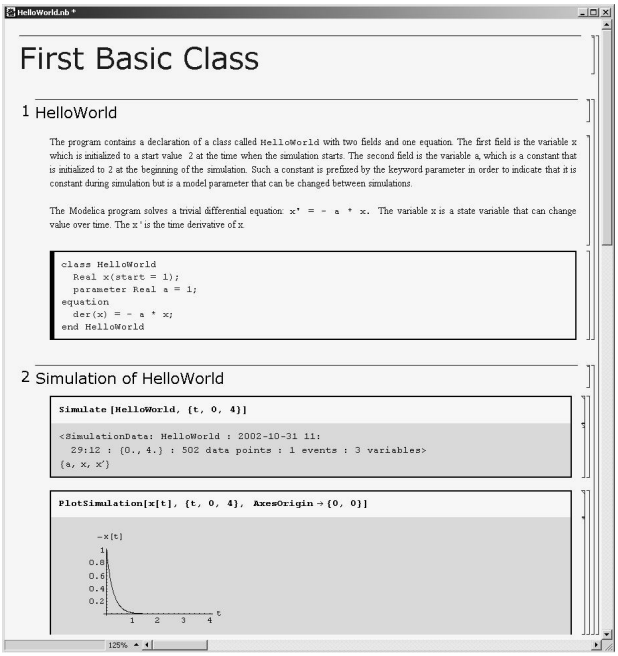


Figure 4. Hello World class.

5. DrModelica on the Web

DrModelica is based on the MathModelica environment. However, MathModelica is an expensive product for universities and other teaching facilities. Making DrModelica available on the web would increase the possibilities to use it, thereby increasing the use of Modelica. This constitutes the motivation for creating a web version of DrModelica.

A requirement of the web version is to only use free and open-source software. Another requirement is to offer a similar structure and similar functionalities as in the MathModelica version of DrModelica. Implementing DrModelica on the web implies that it

is easy to update the material. This way the user does not have to download updates of the material. The interface for the web version of DrModelica is currently available at http://pelix.ida.liu.se/drmodelica/DrModelica.pike.

We had several ideas of how to implement the web environment, for example extending an existing XML-editor or using webMathematica [16]. Exploring these ideas resulted in the conclusion that it would be more work to extend existing tools or applications than to develop a completely new environment.

In order to create the web version of DrModelica a compiler and a user interface are needed. Currently, members of the Programming Environment Laboratory (PELAB) at Linköpings university are working on an open-source Modelica compiler [8], which fits the above-mentioned requirements.

To create the user interface for the web pages of DrModelica a programming language, which can be used to develop the desired functionality, is needed. The interface should have the same functionality as in the MathModelica version of DrModelica.

5.1. Pike

The DrModelica web page uses Pike to generate HTML code. Pike is an imperative, object-oriented programming language with scripting facilities. The language provides multiple inheritance and data structures like arrays and a special kind of array called mapping. Fredrik Hübinette started the development of the Pike language. The first years the development was funded by Roxen Internet Software. Currently, Pike is being further developed by researchers at the Department of Computer and Information Science (IDA) at Linköping University. Several free web servers, for example Roxen WebServer, are written in Pike. At PELAB [17], Pike is used in software composition technology [18] and language connector related research [19].

Reasons for choosing Pike instead of another scripting programming language, like Java Script or Python, is that the time and effort needed to solve the problem would be more effective. Since some of the functionality needed for the web site already exists in Pike, e.g. the ability to expand and compress sections, less functionality needs to be implemented. Additionally, the existing Pike knowledge available at PELAB and the prior knowledge of the developers constitute other reasons for using Pike.

Moreover, Pike fits the requirements for DrModelica on the web. Firstly it is free, which is a necessity. Secondly, it is possible to develop an environment similar to MathModelica using Pike.

5.2. Functionality

In the web version of DrModelica it is possible to create new input cells, where Modelica code can be written (see Figure 5). Each input cell has its own evaluate button for evaluating the Modelica code written in the cell. When adding a new input cell by clicking the button "*Create Input Cell*", the cell will appear in a section that can be expanded or collapsed. A section gets an automatically generated headline, which is currently a number. This number starts at one and is increased for each section that is added. Apart from the input cell and its evaluate button, a section also consists of a text cell, which is used for documenting the code in the input cell. There is no limitation on the number of sections that can be added to the page.

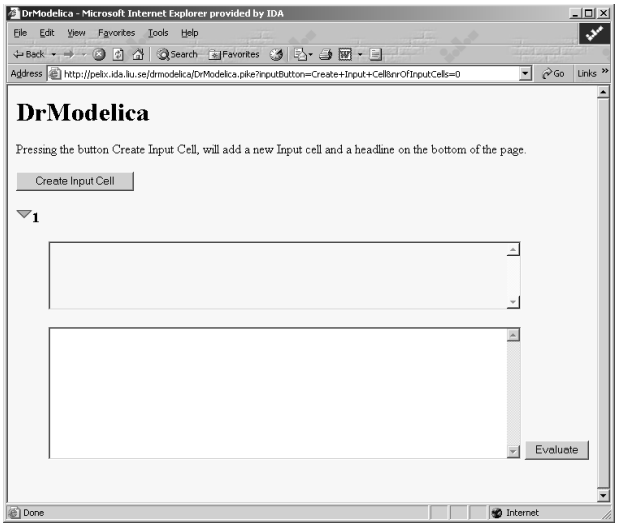


Figure 5. An empty section.

When the evaluate button is clicked, the code in the specific input cell is sent to the Modelica compiler. The result of the execution is presented in a new text area that appears under the evaluated input cell. Figure 6 shows an input cell containing the class HelloWorld. Notice that section 1 is collapsed and section 2 is expanded. The text cell contains an explanatory text, "*HelloWorld solves a trivial differential equation*", about HelloWorld. The class

HelloWorld is transferred to the compiler when the evaluate button is clicked and the result, "*OK*", is shown in the output cell.

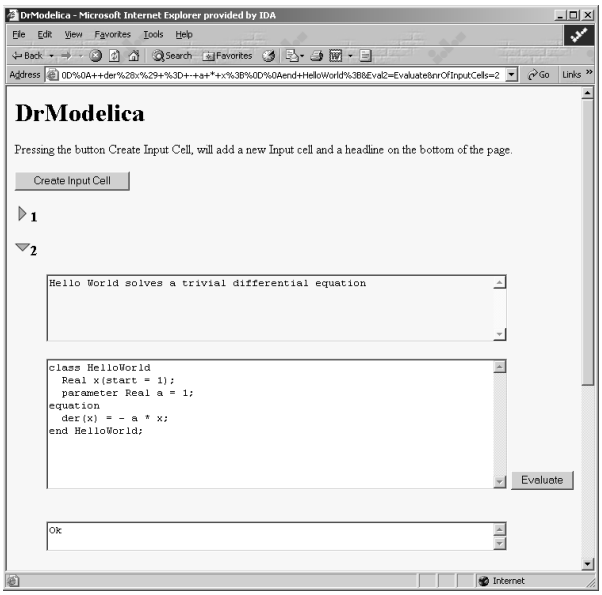


Figure 6. HelloWorld in the web version of DrModelica.

6. Related Work

In the last two decades interactive teaching materials have been developed with the purpose of facilitating the learning process. For example, DrJava and DrScheme are both interactive teaching materials for Java and Scheme respectively. These materials teach the language to the user both by explaining the concepts of the language and by letting the user write programs in a beginner-adjusted environment [20, 21].

6.1. DrScheme

DrScheme [21] is a programming environment for Scheme, providing a graphical user interface, in which it is possible to edit and interactively evaluate Scheme programs. The environment is especially useful for students learning Scheme, since it guides the student through Scheme in a way similar to an introductory course [21].

DrScheme is equipped with three tools to make teaching of functional programming and the development of Scheme programs easier; the symbolic stepper, the syntax checker and the static debugger [21, 22]. The symbolic stepper is a pedagogical tool, which steps forward in the program execution, showing how a program calculates a value. The syntactic checker annotates the source code of a syntactically correct program, by for example changing fonts and colors. MrSpidey is a static debugger [23] and it is the most advanced tool in DrScheme. It marks up potential errors as well as concluding what set of values an expression may produce [21].

6.2. DrJava

DrJava is an open-source, pedagogic programming environment for teaching Java. The environment is influenced by DrScheme, which has served as a model for DrJava [20]. To reduce the problems of learning Java, DrJava first introduces the concepts of coding, as well as testing and debugging the source code, and then focuses on the language.

The tools, used in DrJava, are the Read-Eval-Print Loop, the editor and the integrated compiler. The interface of DrJava consists of two parts, the interactions pane and the definitions pane. The interactions pane provides a Read-Eval-Print Loop, in short REPL, which works as its name suggests: It reads the source code from the interactions pane, evaluates it, and then prints the result back to the interactions pane. When using the REPL no debugger is needed. The same interface is used to run, test and debug programs. Also, as the REPL enables interaction with program components as they are being developed [20].

The compiler, connected with DrJava, is integrated with the source editors, this makes it possible to present the errors from the compiler, only by clicking on them [20].

7. Summary and Conclusions

DrModelica together with previously presented related work have the common goal of teaching a programming language in an environment that has the purpose of facilitating the learning process. DrJava and DrScheme are two interactive teaching materials for Java and Scheme respectively. We have developed DrModelica for teaching Modelica with these two environments in mind. DrScheme and DrJava teach their languages in environments primarily adjusted for beginners by excluding advanced features of the language. While DrScheme and DrJava only focus on a subset of the corresponding language, DrModelica makes the whole Modelica language available to the user.

Learning a new language like Modelica is facilitated by the interactive teaching material. In this paper we have presented an interactive teaching material for Modelica called DrModelica. The idea of this material is that the user shall be able to write, run and test the Modelica code within the same environment.

In order to provide a free and open-source environment for learning Modelica, we created a web version of DrModelica. To our knowledge there exists no interactive teaching material for Modelica and we believe that this material can lead to an increased usage of the Modelica language.

References

[1] Fritzson, P., *Principles of Object-Oriented Modeling and Simulation with Modelica*. 2003. IEEE Press and John Willey.

[2] Grubb, P. and A.T. Armstrong, *Software Maintenance Concepts and Practice (Second Edition)*. 2003. World Scientific Pub Co.

[3] Elmqvist, H., S.E. Mattsson, and M. Otter. *Modelica - A Language for Physical System Modeling, Visualization and Interaction. In Proceedings of the IEEE Symposium on Computer-Aided Control System Design*. 1999. Hawaii, USA.

[4] Fritzson, P. and P. Bunus. *Modelica, a General Object-Oriented Language for Continuous and Discrete-Event System Modeling and Simulation*. *In Proceedings of the 35th Annual Simulation Symposium*. 2002. San Diego, California.

[5] Wolfram Research, *Mathematica*. 4 ed. 1999, Champaign, Illinois. Wolfram Research, Inc.

[6] Fritzson, P., J. Gunnarsson, and M. Jirstrand. *MathModelica - An Extensible Modeling and Simulation Environment with Integrated Graphics and Literate Programming*. *In Proceedings of the 2nd International Modelica Conference*. 2002. Munich Germany.

[7] Jirstrand, M. *MathModelica - A Full System Simulation tool*. *In Proceedings of the 6th Conference on Product Models, Global Product Development*. 2000. Linköping, Sweden.

[8] Fritzson, P., P. Aronsson, P. Bunus, V. Engelson, L. Saldamli, H. Johansson, and A. Karström *The Open Source Modelica Project. In Proceedings of the 2nd International Modelica Conference*. 2002. Munich, Germany.

[9] Elmqvist, H., D. Bruck, and M. Otter, *Dymola - User's Manual*. 1996, Dynasim AB, Research Park Ideon. Lund.

[10] Dynasim AB, *Dymola for Your Complex Simulations*. Available at: http://www.dynasim.se. Last accessed August, 2003.

[11] Fritzson, P., V. Engelson, and J. Gunnarsson. *An Integrated Modelica Environment for Modeling, Documentation and Simulation*. *In Proceedings of the Summer Computer Simulation Conference '98*. 1998. Reno, Nevada, USA.

[12] Nørmark, K. *Requirements for an Elucidative Programming Environment*. *In Proceedings of the International Workshop on Program Comprehension, IWPC'2000*. 2000. Limerick, Ireland.

[13] Ducassé, M. and J. Noyé, *Logic Programming Environments: Dynamic Program Analysis and Debugging.* Journal of Logic Programming, 1994. **19/20**: p. 351-384.

[14] Lengquist Sandelin, E.-L. and S. Monemar, *DrModelica - An Experimental Computer-Based Teaching Material for Modelica*. Master Thesis. *Department of Computer and Information Science*. 2003, Linköping University.

[15] Knuth, D.E., *Literate Programming.* The Computer Journal, 1984. **NO27(2)**: p. 97-111.

[16] Wolfram Research, *What Is webMathematica?* 2002. Available at: http://www.wolfram.com/products/webmathematica/whatis.html. Last accessed August, 2003.

[17] PELAB, *PELAB: The Programming Environments Laboratory*. Available at: http://www.ida.liu.se/~pelab. Last accessed August 2003.

[18] Pike Home Page, *Pike*. Available at: http://pike.ida.liu.se/. Last accessed August, 2003.

[19] RISE Center, *Research Center for Integrational Software Engineering*. Available at: http://www.ida.liu.se/~pelab/inrise/. Last accessed August, 2003.

[20] Allen, E., R. Cartwright, and B. Stoler. *DrJava: A Lightweight Pedagogic Environment for Java*. *In Proceedings of the 33rd ACM Technical Symposium on Computer Science Education (SIGCSE 2002)*. 2002. Northern Kentucky, USA.

[21] Findler, R.B., et al. *DrScheme: A Programming Environment for Scheme*. *A Preliminary Version Appeared at Symposium on Programming Languages: Implementations, Logics, and Programs in 1997*. 2001.

[22] Felleisen, M., et al. *The DrScheme Project: An overview*. *In Proceedings of the ACM SIGPLAN 1998 Conference on Programming Language Design and Implementation (PLDI)*. 1998. Montreal, Canada.

[23] Flanagan, C., *Effective Static Debugging via Componential Set-Based Analysis*. PhD Thesis 1997, Rice University: Houston.