Proceedings
of the 3rd International Modelica Conference,
Linköping, November 3-4, 2003,
Peter Fritzson (editor)

Eva-Lena Lengquist Sandelin, Susanna Monemar, Peter Fritzson, Peter Bunus
PELAB , Linköping University:
DrModelica - An Interactive Tutoring Environment for Modelica
pp. 125-136

Paper presented at the 3rd International Modelica Conference, November 3-4, 2003, Linköpings Universitet, Linköping, Sweden, organized by The Modelica Association and Institutionen för datavetenskap, Linköpings universitet

All papers of this conference can be downloaded from
http://www.Modelica.org/Conference2003/papers.shtml

DrModelica
An Interactive Tutoring 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 through an environment that integrates programming, program documentation and visualization. The teaching material is intended to be used for modeling and simulation related courses at the undergraduate and graduate level.

1. Background

The concepts 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 powerful aids in the product development process. Using advanced tools and languages to build a model of a product and then simulate its behavior, before producing a physical prototype, reduces the number of errors that can occur during fabrication. This reduction 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 hard in order to adapt to new requirements [2].

2. Interactive Environments

Modelica helps solving problems concerning modeling and simulation. In order for Modelica to be used for this purpose, a modeling and simulation environment is needed. In this section the MathModelica environment is presented. MathModelica is partly built on Mathematica technology, which is also described below.

2.1. Mathematica

Mathematica [3] 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 involves using it as a "calculator". The user types a calculation and Mathematica performs it immediately. However, there is a big difference between what a traditional calculator can do and what Mathematica can perform. The system 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, using the `Plot` function.

Mathematica is divided into two distinct parts: the computer algebra engine and interpreter ("kernel") that receives and evaluates all expressions sent to it and the user interface ("front-end"). The front-end provides the programming 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.

2.2. MathModelica

MathModelica, from MathCore Egineering AB [4], is a powerful engineering environment for physical modeling, simulation, analysis and design [5, 6]. In MathModelica, models are described using Modelica. Dymola [7], developed by Dynasim [8], is another powerful Modelica environment.

The MathModelica environment integrates 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 the Model Editor, the Simulation Center and Mathematica notebooks. The Model Editor is a graphical tool for designing models using predefined library components. The Simulation Center is a graphical user interface for running simulations and plotting curves of the models. Mathematica notebooks provide a text based programming environment.

3. 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 [9]. Moreover, it is important to show the result of the source code's execution. In modeling and simulation it is important to have the source code, the documentation about the source code, the execution results 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. Also, users are reluctant to using a programming language that does not provide an adequate programming environment [10]. All the above-mentioned facts constitute our reason for developing DrModelica [11], a teaching material for Modelica. DrModelica is based on MathModelica [4] and the ideas of Literate programming [12].

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 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 provide a subset of the language. Furthermore, we have developed a web version of DrModelica, which

has a similar interface and includes most of the functionality that can be found in MathModelica. The interface for the web version of DrModelica is currently available at http://www.DrModelica.org although in order for the connection between the interface and the Modelica compiler to work, the OpenModelica compiler has to be downloaded first. The difference between the web version and the MathModelica version of DrModelica is that the functionality of the web version is limited, for example there is no possibility to show plots of a simulated model.

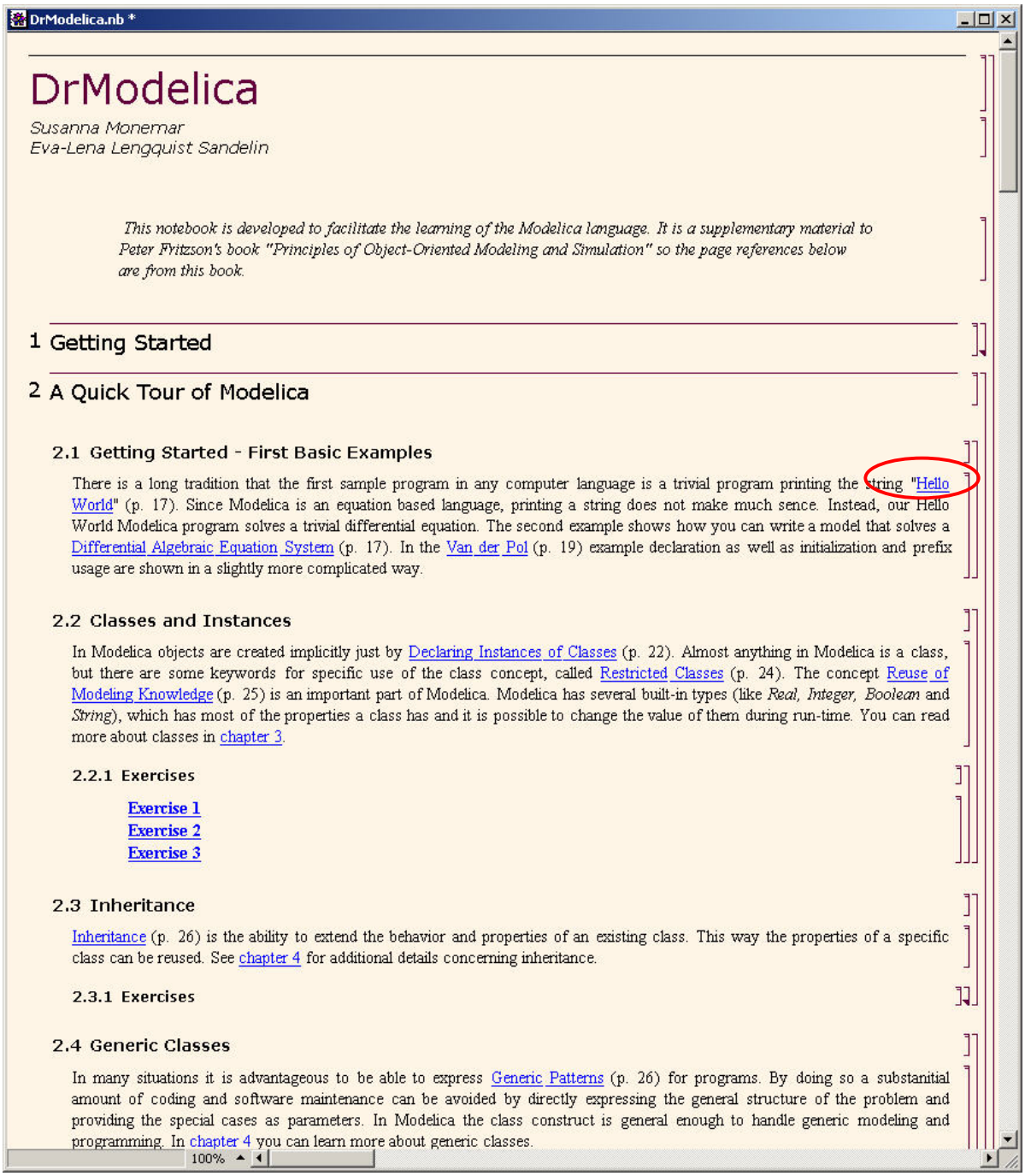

Figure 1. The front-page notebook of DrModelica.

Furthermore, the web version is intended to be used as a testing environment for evaluating Modelica code. It is not a teaching material, since there is no text or examples that the user can learn from.

DrModelica has a hierarchical structure represented as Mathematica notebooks. The front-page notebook is similar to a table of contents that holds all other notebooks together by providing links to them. This particular notebook is the first page the user will see (Figure 1).

In each chapter of DrModelica the user is presented a short summary of the corresponding chapter of 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 user to another notebook describing the keyword in detail.

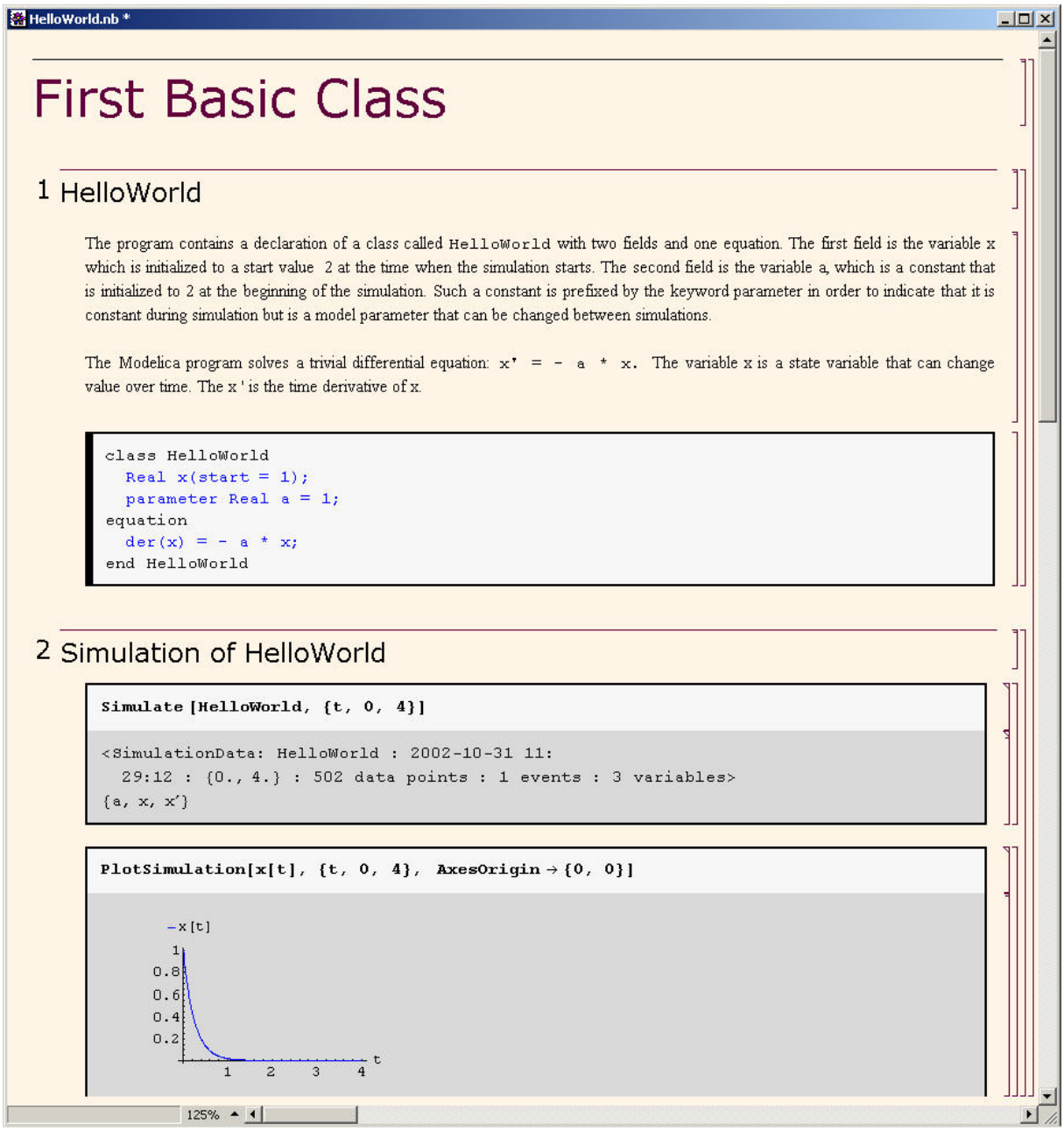

Figure 2. HelloWorld class.

Now, let us consider that the link "*HelloWorld*" in section 2.1 in Figure 1 is clicked by the user. The new notebook, to which the user is being linked (see Figure 2), 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.

No information in a notebook is fixed, which implies that the user can add, change or remove anything in a notebook. Alternatively, the user can create an entirely new notebook in order to write his/her own programs or copy examples from other notebooks. This new notebook can be linked from existing notebooks.

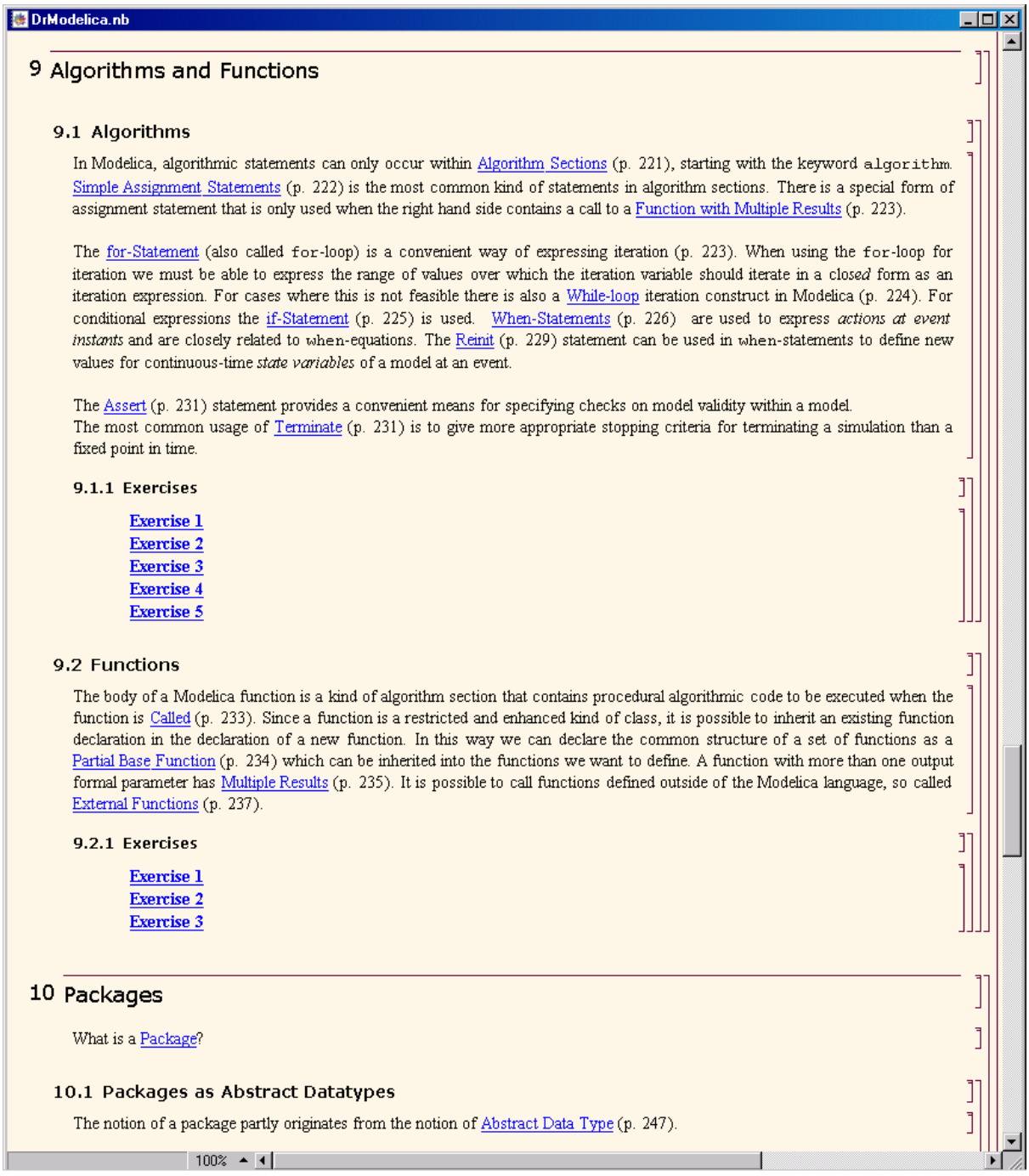

Figure 3. Chapter 9 in the main page of DrModelica.

When a class has been successfully evaluated the user can simulate and plot the result. These two actions are performed by the Mathematica commands `Simulate` and `PlotSimulation`. `Simulate` compiles the code and `PlotSimulation` shows a diagram of the result. Figure 2 shows how `HelloWorld` uses the Mathematica commands `Simulate` and `PlotSimulation`.

After reading a chapter in DrModelica the user can immediately practice the newly acquired information by doing the exercises that concern the specific chapter. We have written the exercises in order to elucidate language constructs step by step based on the pedagogical assumption that a student learns better "*using the strategy of learning by doing*". The exercises consist of either theoretical questions or practical programming assignments. All exercises provide answers in order to give the user immediate feedback.

Figure 3 shows Chapter 9 in the teaching material. Here, the user can read about language constructs, like `algorithm` sections, `when`-statements and `reinit` and then practice by solving the exercises corresponding to the recently read section.

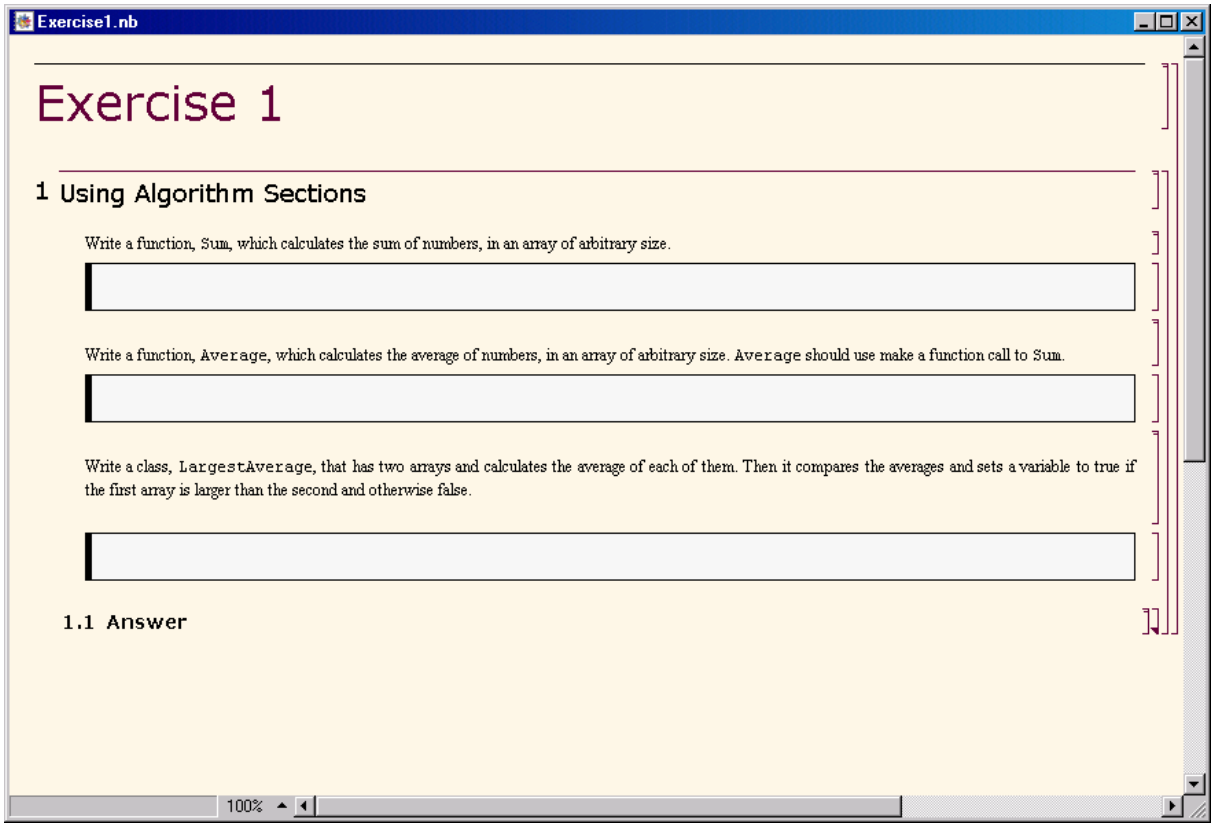

Figure 4. Exercise 1 in chapter 9.

Exercise 1 in section 9.1.1 is shown in Figure 4. In this exercise the user has the opportunity to practice different language constructs and then compare the solution to the answer for the exercise. Notice that the answer is not visible until the *Answer* section is expanded. The answer is shown in Figure 5.

Figure 6 shows that circuits created in the Model Editor of MathModelica can be inserted in DrModelica as pictures and it can be used to generate Modelica code from.

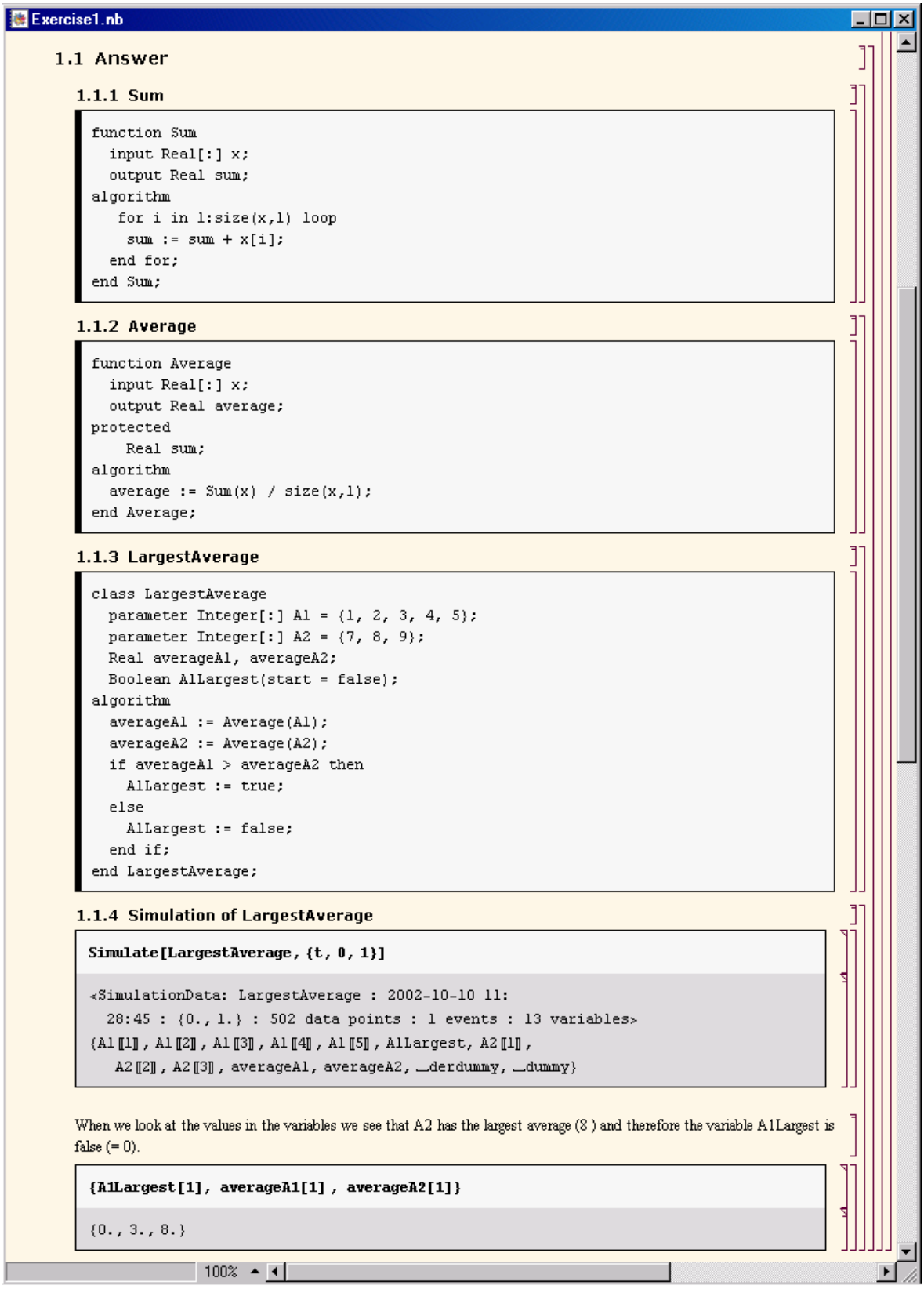

Figure 5. The answer section to Exercise 1 in chapter 9.

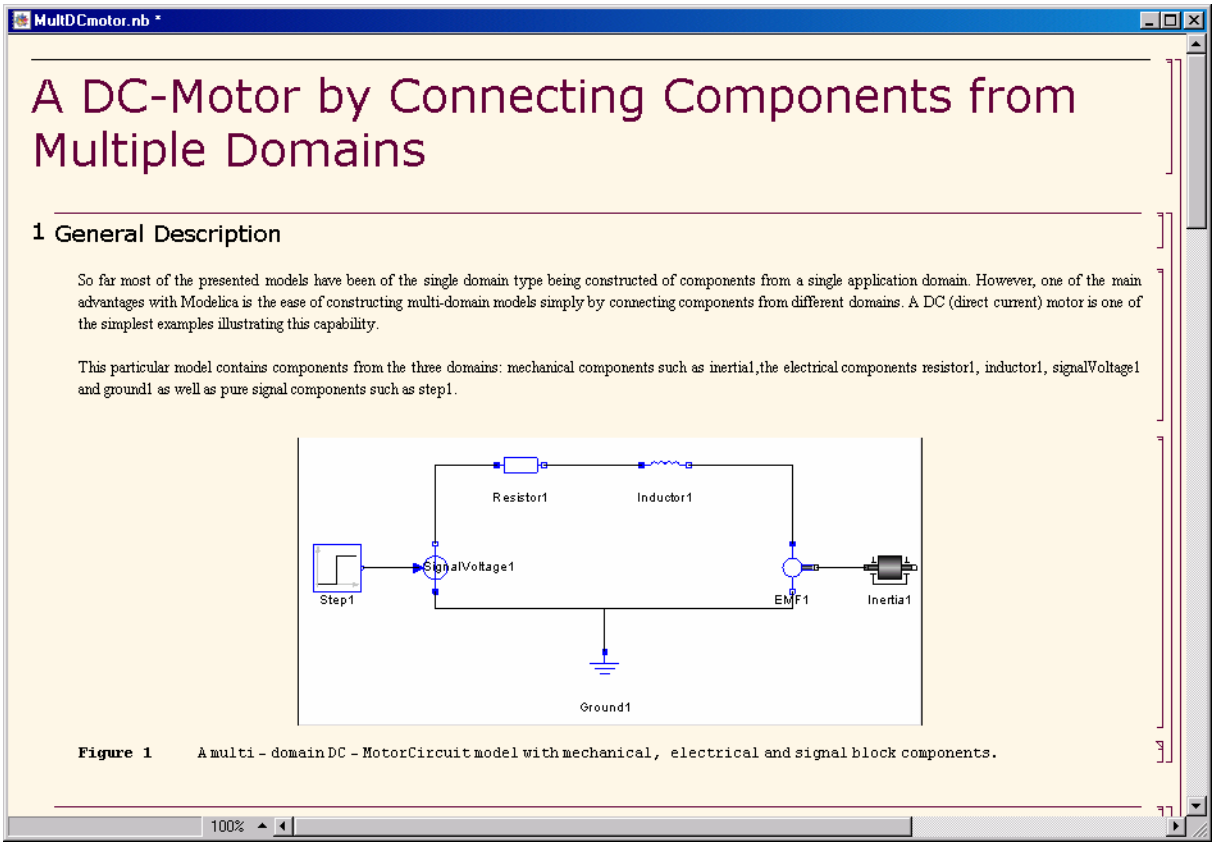

Figure 6. Pictures from the Model Editor in MathModelica can be inserted in the environment.

4. Related Work

During 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 [13, 14].

DrScheme [14] 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 [14].

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 [13]. To facilitate the learning of Java, DrJava first introduces the concepts of coding, as well as testing and debugging the source code, and then focuses on the language semantics.

5. Evaluation of DrModelica

Evaluation methods are important tools for user interface design. Such methods can be divided into usability testing methods and usability inspection methods. The difference between them is that users are involved in usability testing methods but are not involved in usability inspection methods. For evaluation of DrModelica, both methods have been used, with specially developed questionnaires [15] and performing a heuristic evaluation [16].

Using a questionnaire is a usability testing method and reflects the users' subjective opinions. It is a cheap method for testing a system and can be distributed to many users.

Heuristic evaluation is a usability inspection method, which is performed by an evaluator, using a checklist

of guidelines to determine the usability of the user interface. This method is easy to learn and inexpensive to perform. Most of the general usability problems can be identified using a heuristic evaluation. The method requires some experience with heuristic evaluation principles for an optimal result. However, even a non-expert can find many usability problems using a heuristic evaluation.

5.1. Evaluation using Questionnaire

Twelve students attending a graduate Modelica course at Linköping University tested DrModelica. After a few weeks they were asked to answer a questionnaire. All testers were engineering students, either in the area of physics or computer science. The questions in the questionnaire concerned their expectations of the teaching material and if their expectations were fulfilled, what they felt about the approach using literate programming and the structure and layout of the material. The results of the questionnaire were positive. For example, Literate programming was appreciated when programming Modelica. The test group generally found DrModelica to be a better way of learning a programming language, compared to the way they were used to.

The structure of DrModelica and the way of navigating between the notebooks was, according to the test group, fairly easy. The exercises at the end of each chapter were also appreciated by the students. In this way the student was able to “directly use the collected knowledge”, referring to one of the testers.

5.2. Heuristic Evaluation

Three usability experts from HCS (Human Centered Systems), at the Department of Computer and Information Science (IDA) have performed a heuristic evaluation on DrModelica. When performing the evaluation, the evaluators used the guidelines from “Ten Usability Heuristics” [17]. They are listed below:

1. Visibility of system status: The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
2. Match between system and the real world: The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.
3. User control and freedom: Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.
4. Consistency and standards: Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.
5. Error prevention: Even better than good error messages is a careful design which prevents a problem from occurring in the first place.
6. Recognition rather than recall: Make objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.
7. Flexibility and efficiency of use: Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.
8. Aesthetic and minimalist design: Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.
9. Help users recognize, diagnose, and recover from errors: Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
10. Help and documentation: Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

The evaluation gave many valuable results. The evaluators found that learning how to use DrModelica was easy in general. However, realizing how some of the functionality works was, according to the

evaluators, not so intuitive. For example it can be hard to discover the ability to collapse and expand sections. Though, once it was known how to use the functionality they found easy. Furthermore, according to the evaluators it might be confusing that a link in some cases opens a new window and in other cases refers to another chapter in the same window. This is a problem concerning heuristic number 4. Another problem, when being linked to another page, is that there is no feedback telling the user that a new page has appeared in front of the previous one. This is a problem mostly concerning heuristics number 1, 2 and 3. When a new window is opened in front of the other the user is not properly informed about what is going on, since there is no feedback that the window was just being opened (see heuristic number 1). This involves another problem, taking the user back to the former window. This is currently resolved by closing the window, but it would be better solved by having a "back"-button, following real-world conventions (see heuristics 2 and 3). Heuristics number 5, 8 and 9 concern dialogues and error messages, none of which exist in neither DrModelica nor MathModelica, but that is why the environment does not have a need for it. Heuristic number 10 concerns help and documentation. There is a help section on how to start using DrModelica, which was appreciated by the users.

The evaluators also found that DrModelica was less intimidating than other programming environments, since the user is presented with an environment similar to a document showing only a small amount of functionality. This leads the user to believe that DrModelica is a reading material. However, after using the material for a while the user discovers that DrModelica could be used for programming as well. A common approach adopted by many programming environments is to lead the user in the opposite direction, by presenting all functionality from the beginning. This approach can have a discouraging effect on the user.

6. Future Improvements

Considering the results of the evaluation and comparing our work with related work we have discovered some possible improvements that can be implemented in the future. Here follows a list of these improvements:

A suggestion from the students, attending the Modelica graduate course, is to extend DrModelica to contain more exercises on simple as well as more complex constructs in order for the student to get more practice.

Since it can be difficult to learn how to use the functionality in DrModelica, an idea is to make an introductory exercise for practicing the basics step by step instead of just reading a long introductory text.

Links between files containing different variants of the same term should be added.

Currently the exercises in the material mainly concern language specific constructs, it would be desirable to add exercises reflecting the purpose of Modelica. The material needs to be extended with more exercises in general.

Features, like parenthesis matching and keyword highlighting, used in DrScheme and DrJava, would be helpful when programming.

7. Summary and Conclusions

In this paper we have presented the interactive teaching material for Modelica, based on MathModelica, called DrModelica. DrModelica has the goal of teaching Modelica in an environment that has the purpose of facilitating the learning process of the language. Because of the complexity of learning Modelica there is a need for such a material.

DrModelica is based on Literate programming, which enables the user to write, document and execute the source code in the same file or entity. This file or entity becomes a Literate program. In DrModelica the documentation about the source code is not embedded as comments in the code, but instead separated from the code in specific sections only with the purpose of containing text.

The Literate programming approach is extended in DrModelica, in such a way that the result of the executed Modelica program is included in the same file or entity. The results of the source code can be shown in the form of diagrams. This is a necessary part of DrModelica, since Modelica is a programming language used for creating models of complex physical systems and there is a need to check if these models' behaviour follows the specification or comply with the user intent.

The evaluations of DrModelica resulted in many valuable opinions. The members of the test group, answering the questionnaire, generally found DrModelica to be a better way of learning a programming language compared to ways they are used to. One conclusion that can be drawn from the evaluation is that DrModelica is a good teaching material for Modelica. The evaluators also found that Literate programming is a methodology suitable for learning Modelica. DrModelica is developed with the programming environments DrJava (for Java) and DrScheme (for Scheme) in mind.

There is a need for a programming environment for Modelica and DrModelica will hopefully fill this need and increase the usage of Modelica by facilitating the learning process.

The interested reader can visit: http://www.DrModelica.org, where a short version of DrModelica is freely available for download. The full version of the material is included in the software MathModelica and in "*Principles of Object-Oriented Modeling and Simulation with Modelica*" by Peter Fritzson.

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

[4] 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.

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

[6] Fritzson, P., et al. *The Open Source Modelica Project. In Preceedings of the 2:nd International Modelica Conference*. 2002. Munich, Germany.

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

[8] The Dynasim Home Page, *Dymola for Your Complex Simulations*. Available at: http://www.dynasim.se. Last accessed August, 2003.

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

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

[11] 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, Sweden.

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

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

[14] 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.

[15] Nielsen, J., *Usability Engineering*. 1993, San Diego: Academic Press Inc.

[16] Nielsen, J. and R.L. Mack, *Usability Inspection Methods*. 1994: John Wiley and sons inc.

[17] Nielsen, J., *Ten Usability Heuristics*. 1994. Available at: http://www.useit.com/papers/heuristic/heuristic_list.html. Last accessed September 2003.