An Intelligent Tutoring System for Interactive Learning of Data Structures*

Rafael del Vado Vírseda, Pablo Fernández, Salvador Muñoz, and Antonio Murillo

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Spain
rdelvado@sip.ucm.es,
{pablo.fdez.p,salva.ms,murillo925}@gmail.com

Abstract. The high level of abstraction necessary to teach *data structures* and *algorithmic schemes* has been more than a hindrance to students. In order to make a proper approach to this issue, we have developed and implemented during the last years, at the Computer Science Department of the Complutense University of Madrid, an innovative *intelligent tutoring system* for the interactive learning of data structures according to the new guidelines of the *European Higher Education Area*. In this paper, we present the main contributions to the design of this intelligent tutoring system. In the first place, we describe the tool called *Vedya* for the visualization of data structures and algorithmic schemes. In the second place, the *Maude* system to execute the algebraic specifications of abstract data types using the *Eclipse* system, by which it is possible to study from the more abstract level of a software specification up to its specific implementation in *Java*, thereby allowing the students a self-learning process. Finally, we describe the *Vedya Professor* module, designed to allow teachers to monitor the whole educational process of the students.

1 Introduction

The study of *data structures* and *algorithmic schemes* constitutes one of the essential aspects of the academic formation of every student in Computational Science. Nevertheless, the high level of abstraction necessary to teach these topics occasionally hinders its understanding to students. In order to make a proper approach to this issue, we have developed and implemented during the last years, at the Computer Science Department of the Complutense University of Madrid, an innovative interactive and visual learning framework according to the new guidelines of the *European Higher Education Area* and the teaching model focused on the student.

Our innovative approach is based on an *Intelligent Tutoring System* [8] (shortly, ITS), a computer system that provides direct customized instruction

* This work has been partially supported by the Spanish National Projects FAST-STAMP (TIN2008-06622-C03-01), MERIT-FORMS (TIN2005-09027-C03-03) and PROMESAS-CAM (S-0505/TIC/0407).

G. Allen et al. (Eds.): ICCS 2009, Part II, LNCS 5545, pp. 53–62, 2009.

and feedback to our students, a personal training assistant, and a range of tutoring techniques according to the student's response without the intervention of human beings. Thus, our ITS implements the underlying theory of *Abstract Data Types* [5] by doing and enable students to practice their skills by carrying out tasks within highly interactive learning environments. Based on these learner tools, the ITS tailors instructional strategies, providing explanations, hints, examples, demonstrations, and practice problems on data structures and algorithmic schemes [1,4,7]. The evaluation of our research on that systems indicates that students taught by our ITS generally learn faster and translate the learning into improved performance better that classroom-trained participants.

Despite the concept of ITS has been pursued for more that thirty years by researches in education, psychology, and artificial intelligence, few systems are in practical use today for the interactive learning in Computational Science. In order to remedy this lack, this paper describes the design of an ITS which guides the interactive learning of data structures from the *algebraic specification* to the real implementation [5]. The main components of this system are, on the one hand, the *Vedya tool* [9], a visualization tool by means of which, it is possible to provide the students with a complete learning system of both, the main data structures and the more relevant algorithmic schemes. On the other hand, the *Maude system* [3] for the execution of algebraic specifications of abstract data types using the language of formal specification provided by this system. And third, thanks to the development environment of *Eclipse*, we have obtained a fully complete system that is useful for the students as well as the professors, that allows to go from the most abstract level of data structures, provided by its algebraic specification in *Maude*, until its specific implementation in a modern programming language as happens with *Java*. All this learning process can be guided and overseen in a completely autonomous way by using the ITS presented in this paper, through which it is possible to make enquiries about the documentation related to each of the algebraic specifications, to distinguish between the behavior of the data structure and its different implementations through the use of different views or to browse information regarding the cost of the different implementations that have been proposed.

2 The Vedya Tool

Vedya is an integrated interactive environment for learning data structures and algorithmic schemes presented for the first time in [9]. It covers the most common data structures: Stacks, queues, binary search trees, AVL trees, priority queues, and sorted and hash tables. Moreover, it also provides other different types of abstract data types, like one for an implementation of a "doctor's office". Concerning the algorithmic schemes, it covers the most common resolution methods [1,4,7]: Divide and conquer, dynamic programming, backtracking, and branch and bound. All data structures and algorithmic schemes taught in the related study courses are thereby integrated in the same environment.

Currently, there are two versions of the *Vedya* tool. The first version contains all the data structures and algorithmic schemes mentioned above while the new

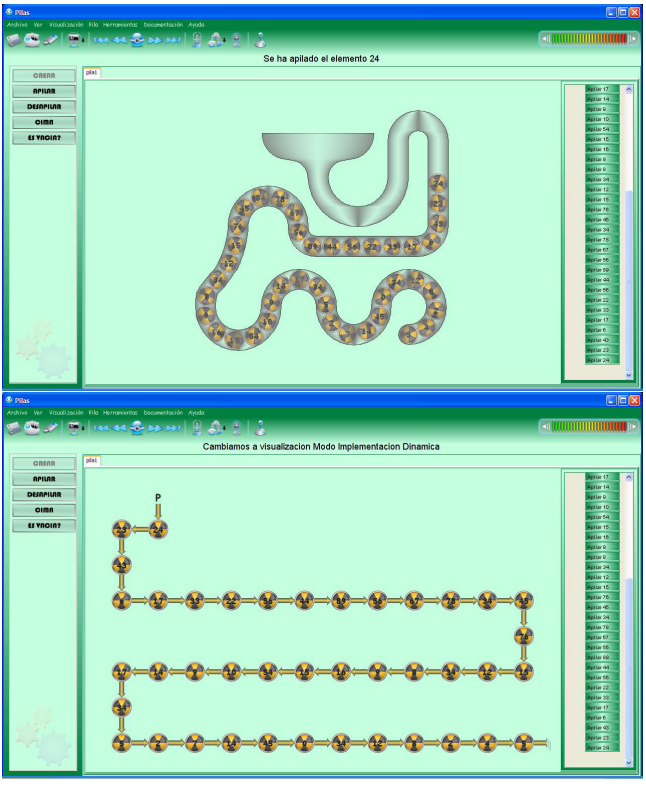

Fig. 1. Data structures in the Vedya tool

one offers a subset of them in a more attractive visual environment. This last version can be found at `http://www.fdi.ucm.es/profesor/rdelvado/Vedya.zip`.

There are several options to use this tool. The main one is the interactive execution, but it is also possible to create simulations that are automatically executed, to visualize tutorials and to solve tests within the same environment. It also integrates a set of animations that show how data structures are used to solve certain problems. For instance, Fig. 1 shows an example of the main windows for stacks. The central panel is used to represent the structure. On the left, there is a list of the actions that can be executed. Partial non-allowed actions are disabled. The right panel shows the visualization of the actions that have been already executed. There are two types of views: The one of data structure behavior to intuitively comprehend its operation, and one or several implementation views, either static or dynamic. On Fig. 1 we show the specific behavior view of a stack and a dynamic implementation based on pointers. The environment also provides documentation about algebraic specification, the implementation code and the cost of each implementation. Moreover, the current version of *Vedya* offers the *Vedya-Test* tool to solve tests (see Fig. 2). This tool can be independently executed and

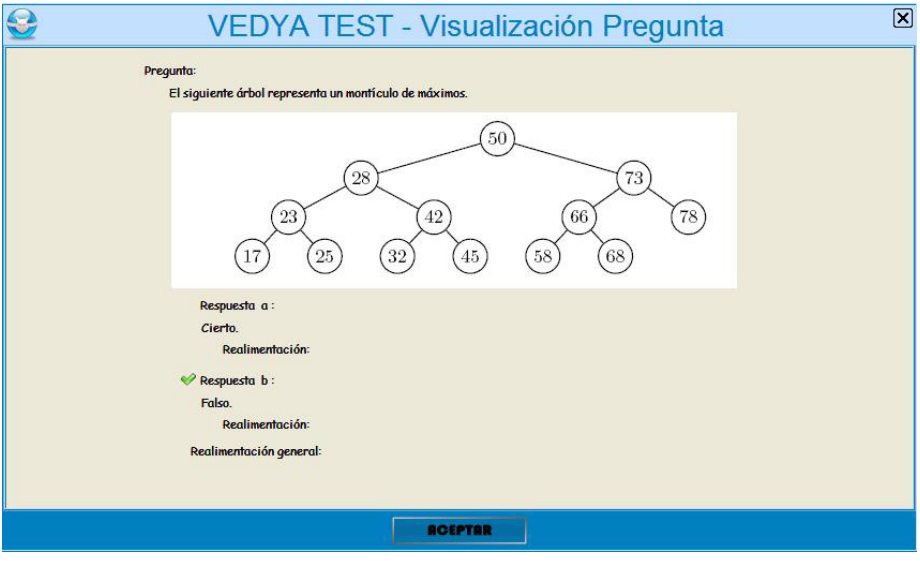

Fig. 2. The Vedya-Test tool for the student evaluation

allows teachers to create, modify or delete questions in a database. The student visualizes the tests, solves them and obtains the correct solutions. Questions are grouped by subject-matter on the database, but it is possible to mix questions about different data structures in the same test. The last version of this tool can be also found at `http://www.fdi.ucm.es/profesor/rdelvado/vedya-test.zip`.

3 Execution of Algebraic Specifications in Maude

For the execution of algebraic specifications in our ITS, we use the language *Maude* [3] based on *rewriting logic*. *Maude* is a high-level language and high-performance system supporting both equational and rewriting computation for a wide range of applications. *Maude* and its formal tool environment can be used in three mutually reinforcing ways: as a declarative programming language, as an executable formal specification language, and as a formal verification system. Moreover, [6] describes the equational specification of the data structures included in the *Vedya* tool now in *Maude* syntax (stacks, queues, lists, binary and search trees, AVL and 2-3-4 trees). The language is available for *Linux* and *Mac-OS* at `http://maude.cs.uiuc.edu`, and there are also extensions for its execution in *Windows* at `http://moment.dsic.upv.es`.

The algebraic specifications can be efficiently executed in the *Eclipse* system (`http://www.eclipse.org/`) by means of special "plugins" (which can be downloaded from `http://www.fdi.ucm.es/profesor/rdelvado/plugins-eclipse.zip`) developed in the Department of Information Systems and Computation of the Technical University of Valencia (DISC-UPV) and in the Computational Languages and Sciences Department of the University of Málaga (DLCC-UMA).

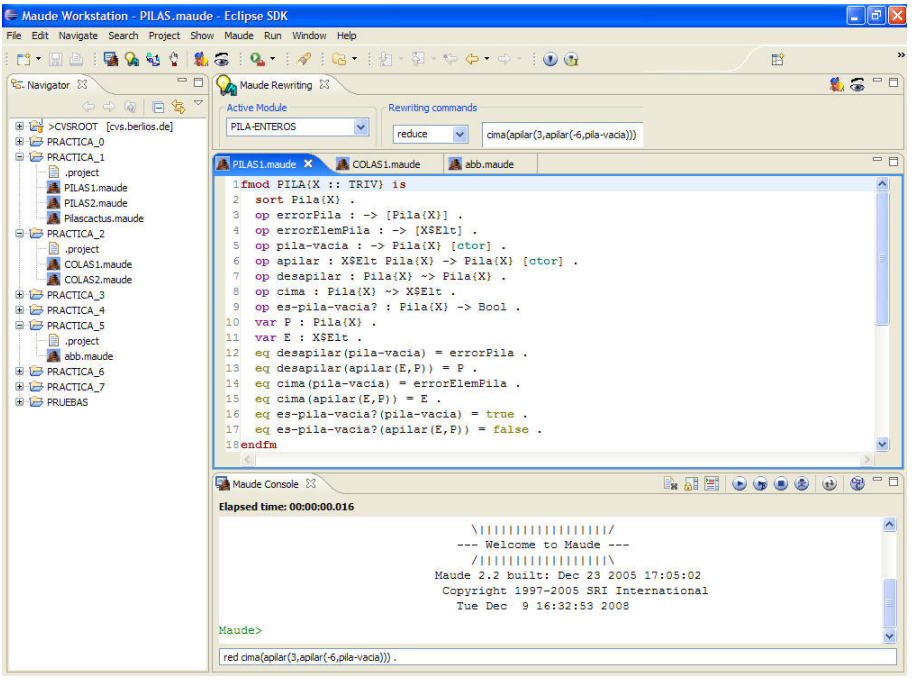

Fig. 3. Integration of Maude in Eclipse for the execution of algebraic specifications

This environment, as shown in Fig. 3, facilitates the student its usage by integrating the text editor with the execution commands of the system. On the left, there appear the developed projects; the central part shows the editor and the execution panel of the system is on it; on the inferior part, the control panel that shows the result of the action.

The basic element of a specification in *Maude* is a "module". The language allows defining the functional modules used to define data types. For example, the functional module for stacks used in Fig. 3 is showed with more detail in Fig. 4. The modules can be customized, using "theories" to such end in order to define the parameters and "views" to relate the formal parameter to the real parameter. The system has predefined the abstract data types most commonly used, as well as the most common theories and views:

```
view Int from TRIV to INT is
  sort Elt to Int .
endv
```

```
fmod STACK-INTEGERS is
  including STACK{vInt} .
endfm
```

In order to execute the specification, the student enters the text given in Fig. 4 in the editor of *Eclipse* (see Fig. 3); then, she/he executes the *Maude* system using

```
fmod STACK{X :: TRIV} is
     sort Stack{X} .
     op   error       :                -> Stack{X} .
     op   error       :                -> X$Elt .
     op   empty       :                -> Stack{X} .
     op   push        : X$Elt Stack{X} -> Stack{X} .
     op   pop         : Stack{X}       -> Stack{X} .
     op   top         : Stack{X}       -> X$Elt .
     op   isEmpty?    : Stack{X}       -> Bool .
     var  P           : Stack{X} .
     var  E           : X$Elt .
     eq   pop(empty)          = error .
     eq   pop(push(E,P))      = P .
     eq   top(empty)          = error .
     eq   top(push(E,P))      = E .
     eq   isEmpty?(empty)     = true .
     eq   isEmpty?(push(E,P)) = false .
endfm
```

Fig. 4. Algebraic specification of parametric stacks in Maude syntax

the existing buttons in the *Maude Console* of *Eclipse* and enters the module. The system detects existing syntax errors and shows them on the *Maude Console*. Once the module shows no more errors, the student may reduce terms by using the equations of the module. To such end, the student may use the commands chart placed at the top of the screen or she/he may directly write the command in the editor and enter it into the system. For example, in order to obtain the top of a stack, we can reduce the term: `red top(push(push(empty,5),4)).` This term must be reduced over the module of the stacks using the integer number theory `INT`. In our example, this module is named: `STACK-INTEGERS`.

The possibility of reducing terms, in an automatic way, allows the students to carry out an initial test of their specifications by detecting many of the errors committed when defining the operations using equations. Another greater advantage of executing the specifications is that the student comprehends the difference between the parameterized module and the instantiated module by being able to reduce terms on different modules. For example, a new module could be named `STACK-CHARACTERS` on which terms of type `red top(push(push(empty, 'a'),'c')).` can be easily reduced. Other examples of abstract data types, such as a "doctor's office" can be proposed [5]. In all of them, the aim was to define parameterized or instantiated data type with different theories. The practical classes are complemented with different terms that the student must reduce over some type of instantiated modules to prove the specification, as well as proposals to make little changes in some actions or erroneous definitions to detect them (see `http://www.fdi.ucm.es/profesor/rdelvado/codigo-maude.zip`).

Taking into consideration that students from the second year were involved, just a few of the language facilities have been used. In superior courses where students have more knowledge on the subject, a richer language can be used [3]

(e.g., many-sorted equational specifications, order-sorted equational specifications, equational attributes, and membership equational logic specifications).

4 An Intelligent Tutoring System for Data Structures

An *Intelligent Tutoring System* (shortly, ITS) for the *Vedya* tool turns into a pedagogical instrument of high practical interest since it attempts to address the whole self-learning process of the main data structures, from the algebraic specification written in *Maude* until the real implementation written in *Java*, within such a powerful and integrated environment as the *Eclipse* system described in the previous section.

The students have their first contact with the data structures that they are going to study by means of the usage of the ITS on *Vedya*. In order to control the student's self-learning process correctly, an online database has been built in on this tool. This means that now, the user has to be logged before using the *Vedya* tool, in order to oversee he/she evolution properly. For this purpose, the additional module called *Vedya Professor* (which can be obtained from `http://gpd.sip.ucm.es/rafav/`), has been designed to take full advantage of this feature. This tool allows teachers to monitoring the current progress of their students as a whole (see Fig. 5), according to the information stored in the database (tests realized, time spent on each test or most consulted documents

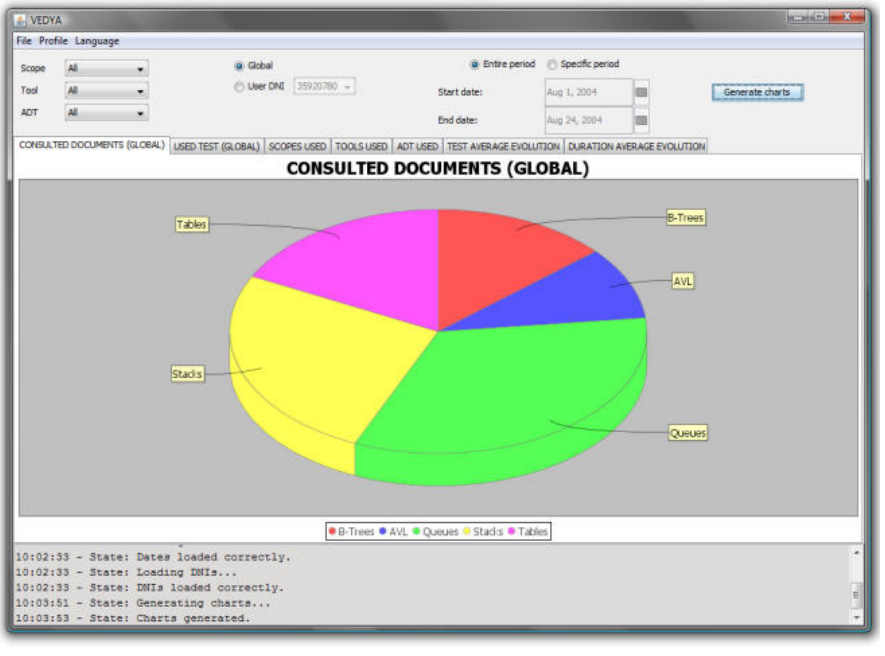

Fig. 5. Vedya Professor Module for the Vedya tool

on the help of *Vedya*). Moreover, this tool also allows seeing detailed information of each specific student, by selecting their identification number.

For example, if their learning of data structures is focused on linear data structures or binary search trees, the ITS would suggest that the student should start their learning process in the corresponding section of the tool, where they will be able to experiment, freely and on their own, each one of the actions offered by these structures (see Fig. 1). In order to strengthen and evaluate this intuitive knowledge, the student has, in addition, the useful possibility of using the *Vedya-Test* tool (see Fig. 2).

Once the student has a clear idea of the informal behavior of the data structure, the ITS may continue working on the *Eclipse* system. The first step would be to formally capture the intuitive knowledge that the student has obtained through the usage of *Vedya* in a specific algebraic specification written in *Maude* syntax. In order to facilitate this difficult step in the student's self-learning, the student may use, interactively, the documentation that is included in the manual of the *Vedya* tool.

Once the algebraic specification in *Maude* syntax (see Fig. 4) is entered into the *Eclipse* system (see Fig. 3), the student can now go on executing little tests using the *Maude Console*, in order to check whether it coincides with the intuitive and informal notion of data structure from which he/she initially departed in *Vedya*. Such experience would allow the student to reach the high level of abstraction that is necessary in computer supported education for each formal specification of a software component, always based on the intuitive and experimental knowledge.

Once the algebraic specification of the data structure is obtained, the next step performed by the ITS would be to develop an implementation in an object-oriented programming language such as *Java*, by means of the facilities provided by the programming environment in *Eclipse*. This time, the student may use the algebraic specification that she/he has built, as if dealing with an authentic "instructions manual". The main advantage of our methodology is that the specification behaves now as a prototype of the data structures to be implemented, in a way that the student is able to find out the exact behavior for all those moments of doubt that may appear during the design process, even before the student is able to compile their programs. In order to be able to guide, in a more specific way, the step from specification to implementation, the student may make use again of the *Vedya* tool. This time, the student may access to the part that would correspond with the implementation of the data structure that she/he is studying from the options menu (see Fig. 1). From there, he/she may try different implementation possibilities based on arrays or pointers.

Once the student is familiar with the different implementations of the structure, she/he is finally ready to properly decide on a suitable representation in the *Java* language. The possibility of having understood and previously evaluated the different implementations by means of the ITS allows the student the possibility to acquire a clear knowledge of the *algorithmic cost* of the chosen implementation in *Java* for each specific operation of the data structure, so that

this would also be a decisive criterion at the moment of designing its own implementations. In this part, the "*algorithmic schemes*" part of the *Vedya* tool plays an important role, since it allows the student to acquire a good programming methodology.

5 Evaluation

In order to obtain a detailed evaluation of the usage of the ITS on *Vedya* and *Maude* in our integrated *Eclipse* system, we have proposed several tests (see `http://www.fdi.ucm.es/profesor/rdelvado/Tests.zip`) related to the behavior, specification, implementation and application of the main data structures offered by the tool in the "Data Structures" academic subject at the second year, and in the "Programming Methodology and Technology" subject at the third year.

Taking into account this profile of our engineering and Computer Science students, we have proposed 8 tests in the *Virtual Campus* of the Complutense University of Madrid (`http://www.ucm.es/campusvirtual/CVUCM/`). The number of engineering students registered in this *Virtual Campus* was just over 122. Fig. 6 shows the number of the students who answered each of the tests. We observe that, from the third test on, the number of students becomes stable in a number lightly low to the number of students who access regularly to the *Virtual Campus*. These numbers, though seemingly high, are only between 30% and 40% of registered students, which shows the high rate of students giving up in this topic from the beginning. Fig. 6 also shows the percentage of correct answers: In general, it is high, which demonstrates the interest of the students who have taken part. Fig. 7 shows the percentage of students that did not attend the final exam, those who passed, and those who failed during the last six years. We observe that in the last academic courses, in which we have applied the ITS on *Vedya* tool, we have reduced by 10% the percentage of students giving up the course with respect to the previous course, and at the same time, we have increased by 12% the percentage of students that passed the exam. The percentage of students that failed the exam decreased by 2%. Comparing with previous

	Stacks 1	**Stacks 2**	**Queues**	**Sequences**	**BST**	**AVL**	**RB**	**Heaps**
Students	65	61	57	31	35	38	32	39
Answers	76.4%	82.5%	77.8%	65.6%	82.2%	84.9%	80.2%	86.3%

Fig. 6. Students answering the tests and percentage of correct answers

	2002/03	**2003/04**	**2004/05**	**2005/06**	**2006/07**	**2007/08**
Not attended	57.6%	45.3%	42.3%	64.7%	50.8%	40.2%
Passed	15.3%	22.2%	20.2%	18.2%	30.1%	42.6%
Failed	27.1%	32.5%	37.5%	17.1%	18.9%	17.2%

Fig. 7. Comparison of academic results with previous courses

courses (2003 to 2004) the percentage of students that passed has increased between 20% (with respect to the course 2003/04) and 25% (with respect to the course 2002/03).

6 Conclusions and Future Work

In the last years, many papers on visualization of data structures have been written (see, e.g., [2]). Nevertheless, there is a lack in many of them of a tutoring system which guides the interactive learning of data structures from the algebraic specification to the real implementation by means of appropriate user tools.

In this paper, we have described the design and usage of an innovative educational environment for the interactive learning of data structure by means of an *Intelligent Tutoring System* [8]. This system can be efficiently applied on the visualization tool *Vedya* [9] and the specification language *Maude* [3] with its programming environment in the *Eclipse* system, allowing the students the possibility of acquiring the capacity of implementing, correctly and properly, a data structure according to its formal algebraic specification, using in their design, the proper algorithmic schemes. As a consequence, it is possible to provide the students with a complete and professional methodology of software development that is very useful in the current teaching of Computer Science.

As future work, we plan to integrate, as part of the development of our intelligent tutoring system, an interface of the current "*Vedya-Maude*" system in *Eclipse*, in order to control and guide students along their self-learning process in a more autonomous way.

References

1. Brassard, G., Bratley, P.: Fundamentals of algorithms. Prentice Hall, Englewood Cliffs (1996)
2. Chen, T., Sobh, T.: A tool for data structure visualization and user-defined algorithm animation. In: Frontiers in Education Conference (2001)
3. Clavel, M.: All About Maude - A High-Performance Logical Framework. LNCS, vol. 4350. Springer, Heidelberg (2007)
4. Cormen, T., Leiserson, C., Rivest, R.: Introduction to Algorithms. The MIT Press, Cambridge (2001)
5. Horowitz, E., Sahni, S., Mehta., D.: Fundamentals of Data Structures in *C++*. W.H. Freeman & Co., New York (1995)
6. Martí, N., Palomino, M., Verdejo, A.: A tutorial on specifying data structures in Maude. Elsevier ENTCS 137(1), 105–132 (2005)
7. Neapolitan, R., Naimpour, K.: Foundations of algorithms using C++ pseudocode. Jones and Bartlett (2003)
8. Psotka, J., Mutter, S.A.: Intelligent Tutoring Systems: Lessons Learned. Lawrence Erlbaum Associates, Mahwah (1988)
9. Segura, C., Pita, I., del Vado, R., Saiz, A., Soler, P.: Interactive Learning of Data Structures and Algorithmic Schemes. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part I. LNCS, vol. 5101, pp. 800–809. Springer, Heidelberg (2008)