An Adaptive Web-Based System for Learning Programming

Mária Bieliková
Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Institute of Informatics and Software Engineering
Ilkovicova 3, 842 16 Bratislava, Slovakia
E-mail: bielik@fiit.stuba.sk

Abstract. This paper presents an approach to learning programming by novice programmers through a web-based adaptive educational system called ALEA (Adaptive LEArning). ALEA supports learning programming by generating sequences of program examples that serve as exercises for a learner. The sequence is adapted to the needs of individual learners. At present, ALEA contains more than a hundred Lisp and Prolog program examples. The program examples are presented as specific instances of program schemata which facilitate understanding of basic programming principles. We have been using the program schemata in teaching an introductory course on functional and logic programming for nearly ten years. Until the ALEA system was developed, the program schemata constituted an important part of the lectures. At present ALEA enables effective acquisition of basic programming skills through usage of program schemata and adapted sequences of program examples. We also discuss our experience in using web-based support for teaching programming.

Keywords: learning programming, program schema, adaptive educational hypermedia, web-based systems

1 Introduction

Adaptation to learners and their activities is becoming one of the major requirements to web-based educational applications. Therefore such applications should possess certain level of intelligence in order to help and guide learners effectively. As a response, most of the well-known technologies from the area of intelligent tutoring systems are being gradually re-implemented for the Web. They are naturally combined with adaptive hypermedia technologies [1].

Web-based adaptive hypermedia applications use several approaches to achieve adaptation. The *adaptation of* the presented *content* or its *layout* is the most visible and therefore the most popular one. Through adapting the visualization of the same content, different users get different output according to their current needs. Content highlighting by using various colours and/or pictures is the most frequently used technique. The adaptation of content involves inserting/removing fragments, altering fragments, sorting or dimming fragments, etc. [2].

Typical adaptation used by the adaptive hypermedia systems is the *adaptation of navigation.* When a user follows a link in a standard web application this results in displaying the requested page. Adaptation featured systems usually include additional processing in link resolution. For example, if a user clicks on a link to display content related to a concept, the system may offer

them a number of pages to read before presenting the requested page (in case they have not read those pages yet). Adaptive navigation can be also used as a generalization of curriculum sequencing technology in a hypermedia context [3], thus providing the most suitable sequence of knowledge units to be learned by the individual learners.

In this paper we present an approach to web-based support for programming skills development, which can only be obtained through practice. Learning programming is a process which obviously requires study of the programming principles together with sufficient practice. The existing web-based textbooks are suitable for study of programming language concepts. Only a few of them, however present adaptive educational applications; in other words, the content and/or navigation can rarely be adapted to a student or to a students' group requirements. The ELM-ART system is an example of a well-known adaptive educational system. It supports learning programming in Lisp. It also provides additional functionalities typical for courseware management systems. The system was first implemented in the 90-ties as a standalone platform-dependent application, and was later rebuilt to a web-based version [4].

The adaptive educational hypermedia application presented in this paper - ALEA (Adaptive LEArning) - is based on the idea of teaching fundamentals of programming by *program schemata* construction and explanation. ALEA provides students with support for learning programming by generating adaptively sequences of programming tasks together with their solutions (called also program examples). ALEA content includes functional and logic program examples [5].

The paper is organized as follows. Section 2 describes our approach to teaching programming using program schemata. Next, we discuss ALEA, a web-based adaptive application supporting acquisition of programming skills. An evaluation of the described approach to learning programming by novices and ALEA usage is presented in Section 4. The paper is concluded by a summary of the work and a brief discussion of future research.

2 Program schemata as a tool for learning

The design of a support tool for learning programming requires an investigation and capturing of the nature of computer programming. This can be done by specifying the relevant knowledge, finding ways how to represent it and inferring its appropriate usage to generate recommendations to a student. The approach to teaching programming we use can be best characterized as an explanation and use of *program schemata*. This was largely inspired by the work of Gegg-Harrison [6,7]. It is also in full agreement with the trends in the Computer Science education towards representing standardized computer programming knowledge as program plans, program schemata, program patterns, or program skeletons [8,7,9,10].

ALEA is a general-purpose educational web-based adaptive system. We use it in teaching functional and logic programming, so presently, its information space contains texts on functional and logic programming and functional and logic program schemata together with program examples. The examples used in this paper are related to the topic of list processing in the functional programming language Lisp.

When tackling different problems, basic program schemata should be used in accordance with various programming techniques. The crucial point is to recognize a program schema and then to specialize appropriately its generic parts in order to solve a problem. This is a particular

know-how that a student should learn. To achieve this, the students learn about the kinds of problems which occur most frequently, and about the program schemata which correspond most closely to those kinds of problems. Another approach which complements the previous one consists of first presenting a program schema to the students and then asking them to practice the schema specializations to solve given problems.

The first approach is based on proceeding *from specific program examples to program schemata* (generalization). The second approach is based on proceeding *from general to specific* (specialization). We found altering these two approaches advantageous for improvement of learning programming effectiveness. As discussed by Navrat in [11] the interplay between generalization and specialization is critically important for the programming activity. Abstraction and concretization are used naturally during programming practice.

We see the learning programming process divided into two parts:

- Students try to understand the essence of the programming paradigm (in this case functional) by learning to recognize correctly a program schema or a combination of program schemata which is to be applied in a given situation. After recognizing the program schema, the task of formulating the required function definition is often quite straightforward. The teacher guides and helps the students in organizing their schema knowledge into a *mental library* which allows a more effective use of program schemata while solving more complex problems. This can be achieved by providing more practice through solving a larger number of simple problems.
- Students then practice programming on more complex problems. In case of functional programming they design and implement an abstract data type (e.g., set, table, array or graph) based on its specifications. Here, we require strict adherence to the functional programming style.

Logic programming is taught by using the same methodology. Naturally, specific features of the logic programming paradigm are considered here, such as backtracking or variable instantiation.

The main advantages of the described approach to teaching programming are as follows [9]:

- Use of program schemata facilitates understanding the basic programming principles.
- Process of acquiring the basic programming skills is speeded up.
- Novices write better programs in the sense that they correspond better to the considered programming paradigm (be it functional or logic). In fact the students do not have much chance to apply their previously acquired procedural habits as these come soon into conflict with the program schemata.
- Program schemata not only provide students with structured programming knowledge, but are also used by the teacher for assessment of the students' mastering of programming.

ALEA aims at supporting the first part of the learning process. It implements adaptive navigation support using two adaptive hypermedia techniques:

- *local guidance*: by presenting "forward" links,
- *local orientation*: by presenting a list of concepts representing the most interesting part of ALEA information hyperspace with regard to the individual learner characteristics; the links are in several colours expressing their suitability to the students' needs.

2.1 Program schemata for functional processing of lists

Students who enrol in the course of Functional and Logic Programming are novices in programming (with regard to the functional and logic programming paradigm). We discuss here only functional programs which are suitable for demonstration purposes. In the beginning several restrictions to the presented functional program examples are applied: (1) the only kind of data are lists including atoms (numbers and symbols); later on dotted-pairs are considered as well; (2) no indirect recursion is involved; (3) no operations with side effects are involved. Other program schemata are presented gradually when proceeding to solving more complex problems (e.g., input/output). Since the list is the basic data structure used in the Lisp programming language, the problems related to list processing are highly suitable for learning functional programming by novice programmers.

A particular program schema describes the typical structure of functions that solve a class of similar problems. We define program schemata in a Lisp-like language [9], using two kinds of variables: first order variables (Lisp function arguments) and program schemata variables (variable function symbols). This language is further simplified in the ALEA system, so that program schemata can be used more effectively by the novice programmers. Only variables significant to the program schema are explicitly presented.

For example, in the program schema for list mapping given below, the `<Map>` and `<Transform>` variables are program schema variables, and `List` is a function argument. Naturally, a function corresponding to this `Map` schema can have more function arguments (in addition to `List`).

```
(defun <Map> (List)
    (cond ((null List) nil)
          (t (cons (<Transform> (first List))
                   (<Map> (rest List))) )   ))
```

In list processing we have identified four basic classes of typical problems – *reduction*, *mapping*, *filtering* and *predicate* (see Fig. 1).

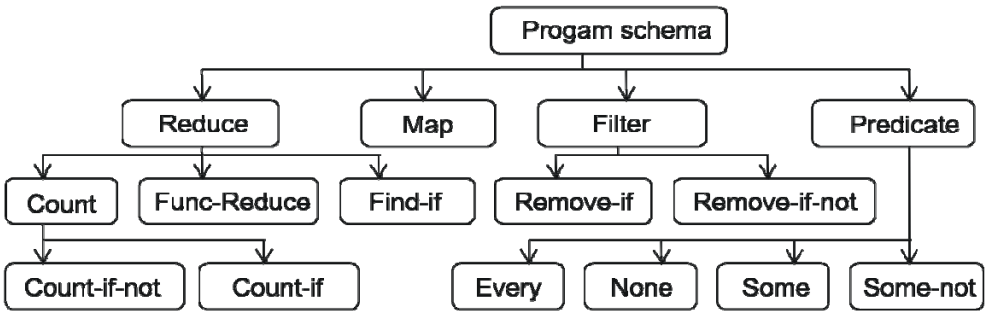

Figure 1 The hierarchy of program schemata for processing of lists

Program schemata used for teaching programming are organized into hierarchies. Students learn linear list processing, where just top level elements of the list are processed and recursive list processing, where lists within the given list are recursively processed. Both groups are characterized by the same classes of problems. However, in most cases of processing of recursive

lists, it is advantageous to consider the argument not to be a list, but an arbitrary s-expression (a binary tree).

Fig. 2 shows the reduction schema (using fat and tail recursion) as it is presented to a student in the ALEA system.

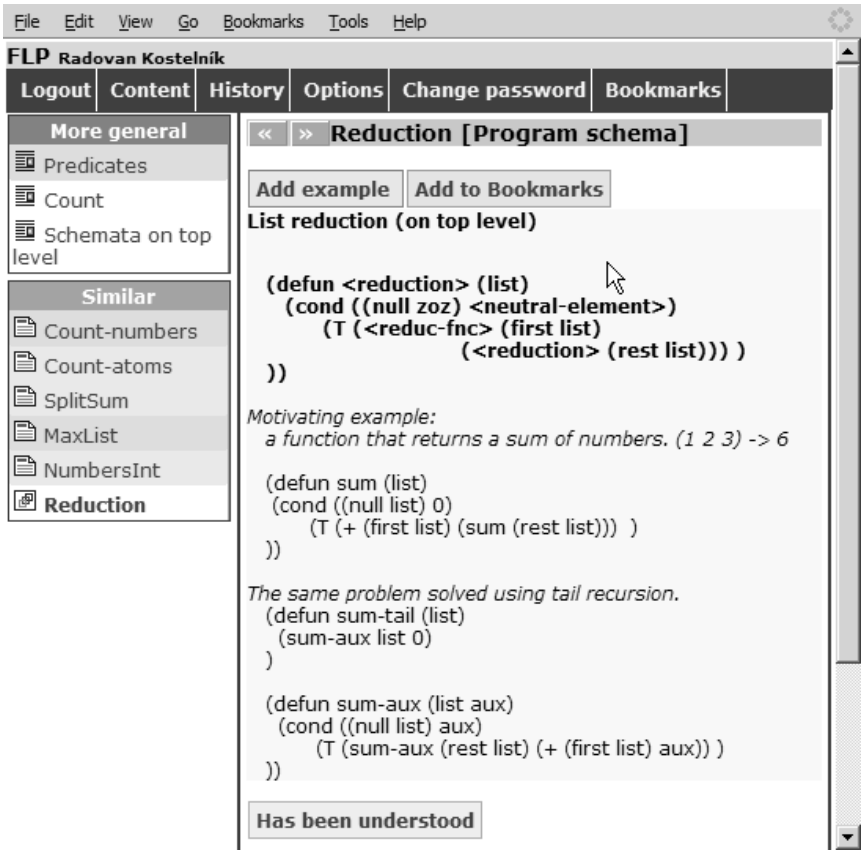

Figure 2. Reduction program schema (the content area of ALEA screen is shown).

2.2 Program examples

In ALEA program schemata are applied by defining a special value for each schema variable and completing additional function arguments. The crucial issue in applying a program schema to solve a problem is the ability to recognize a situation when a combination of program schemata is desirable, for example as in a problem where the elements of a list which fulfil a given condition are to be mapped and the other ones to be simply deleted. The solution in this case is to combine a program schema for a list mapping and a program schema for a filter.

ALEA presents each program example using three information fragments:

- a problem definition,
- hints,
- a solution or several alternative solutions together with explanatory notes.

Since ALEA is aimed at novice programmers, the size of solutions (programs) is in most cases several tens of lines of code. Fig. 3 shows ALEA screen with the solution of a programming task using the program schema for list mapping. The problem is to define a

function taking a list as input and returning a list in which all s-expressions from the input list that are different from a given number are mapped to “1” and all others are mapped to “0”.

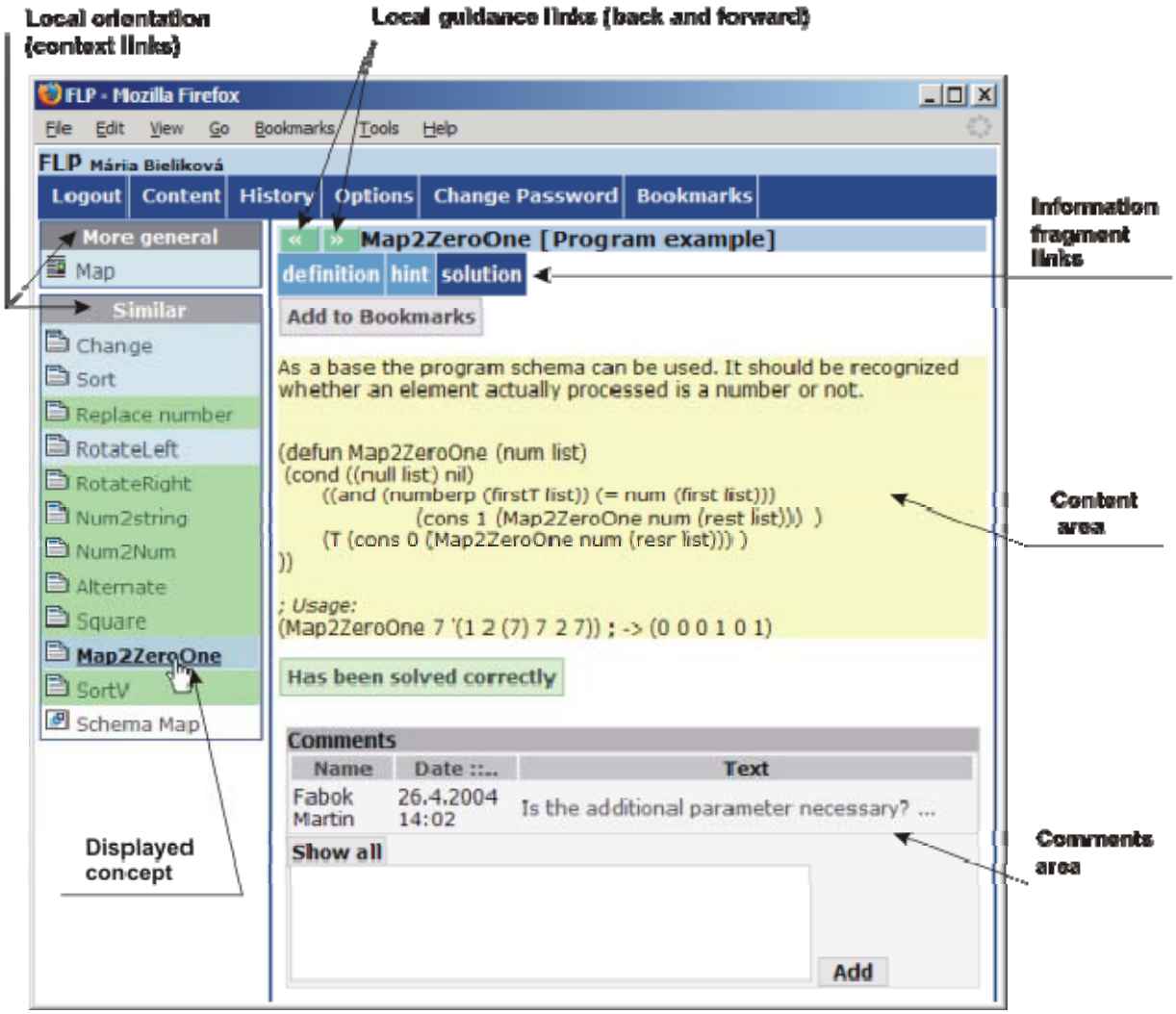

Figure 3. A screen shot from ALEA interface representing a solution of the list mapping programming task.

The left pane of the ALEA screen contains information related to solving the current problem in the form of context links. It is divided into two areas – a list of *more general* concepts and a list of *similar* concepts. The latter are neighbours of the presented concept in the concept hierarchy graph with respect to the *subconcept*, *instance* or *schema* relations. The entries in the lists can be of three kinds representing different information about the concepts: text, program examples and program schemata. Obviously, they correspond to the chosen domain and a change of the domain model will lead to change of the presented types of information about the concepts. The information types are indicated by different icons displayed on the left of the item names. For the learning programming domain the following icons have been chosen:

- a program example,
- a program schema,
- text.

All items in the lists are presented as links providing one-click access to the corresponding information content. Each item has a background of one of three different colours in order to indicate: recommended (green), visited (blue), and learned (white) content.

2.3 Learning sequences

One of the important issues in teaching list processing is the order in which the learners become acquainted with the various program schemata. Pairs of similar problems are defined for both groups of program examples (related to linear and recursive list processing). An example of such pairs of problems is to substitute the first occurrence of a given symbol in a list by another symbol and similarly, to substitute all occurrences of that symbol in the list. Our experience obtained during a number of years is that students can cope better with problems from the former class. Thus ALEA offers in the beginning program examples for practicing linear list processing.

ALEA infers a sequence of information items to be presented to a student who has not mastered sufficiently a specific programming concept (program schema). It increases the number of recommended tasks for such a concept. This approach can be characterized as a directive instruction [12] which is appropriate for novice learners.

The instruction provided by ALEA is performed in the following cycle:

- The system presents an explanatory text, a program schema or a program example.
- The learner responds.
- The system infers the best adaptation for the next step of learning.

The learner responds using local guidance navigational tools (forward and back links to the left from the title of the presented content, see Fig. 3) or indicating the level of comprehension of the presented material (e.g., “Has been solved correctly” button, see Fig. 3).

The application domain of programming gives us an opportunity to lead the learners along different paths through the information base. Some students prefer first to see an explanatory text related to a concept they are learning, then a generalization of the learned programming concept (given by the program schema) and finally to practice programming by solving given tasks. Hints enhance the process of learning. Other students prefer going straight onto solving programming tasks immediately after seeing the explanatory text (if ever), and only then they look at the program schema related to the tasks and compare their solutions to the presented generalization.

The above mentioned sequences represent the common approaches to learning programming. The first one is known as “*from general to concrete*” and the second one “*from concrete to general*”. A selected approach typically determines the order of concepts presented to the learners.

3 Adaptive learning with ALEA

3.1 ALEA models

Typically, the adaptive hypermedia applications include the following key models [13,14]:

- *Domain model* describing the information content structure along with the current content.
- *User/environment model* containing the user or environment specific data often related to the information content.

- *Adaptation model* consisting of a specification of adaptation knowledge (frequently represented using rules) and an adaptation engine (responsible for performing an adaptation based on the adaptation knowledge according to the user/environment model).

The domain model is used to structure the hypermedia content. It consists of concepts and concept relationships [13]. The ALEA domain model exploits a typical representation by a graph in which each node represents a concept and edges represent relations among them. Concepts and relations have additional attributes (e.g., type, visibility, etc.). In accordance with the general hypermedia models ALEA represents and stores the information content (e.g., text, pictures) in the information fragment base separately from the concepts. The relationship between the concepts and the information fragments can be M to N, i.e. one concept can be presented by several information fragments and one information fragment can relate to several concepts.

The concepts in the ALEA domain model represent programming knowledge presented as program schemata and program examples including their definition, hints and solution, as well as other explanation texts. The current information base consists of more than one hundred simple problems for the programming languages Lisp and Prolog.

The application domain is described using the following relations (see an example of a part of the ALEA domain model for functional list processing on Fig. 4):

- *has-subconcept*: relation between two non-specialized concepts (typically explanatory texts),
- *has-instance*: relation between a concept and a program example,
- *has-schema*: relation between a concept and a program schema,
- *similarity*: relation between two or more similar concepts,
- *prerequisite*: relation determining the order in which the concepts are to be learned.

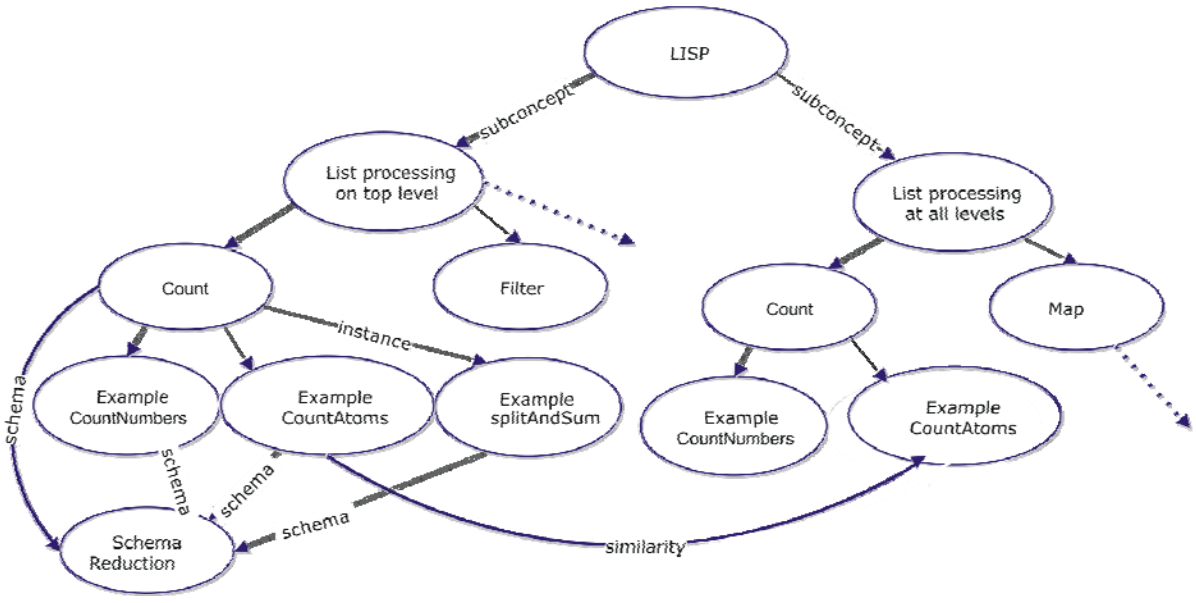

Figure 4. Part of the ALEA domain model for functional list processing.

The ALEA user model is of overlay type [1]. It stores user-specific values for each concept and page or information fragment. For each visited fragment the number of visits to the page displaying the fragment, as well as the date of the last access is stored in the learner model. For

each concept, the estimated level of knowledge of that concept by the learner is stored. The learner's level of knowledge is computed in accordance with rules defined in the adaptation model. Currently, we use a simple heuristic based on the number of visits of particular concepts combined with the user's indication of their level of comprehension (by using a special button displayed in each information fragment, e.g. the "Has been solved correctly" button presented in Fig. 3). User's behaviour is recorded in a sequence of actions (e.g., `ContextClick`, `MarkUnderstood`, `Login`, `Logout`, etc.) which are also used in adaptation.

The main tasks of the adaptation engine are to select the content (including the list of links for local orientation) to be presented to a learner and to prepare it to suit the learner's needs. In ALEA the adaptive content selection knowledge is divided into three layers: (1) learning sequence layer, (2) concept layer and (3) information fragment layer. Each layer is represented separately using condition-action rules, written in XML. The ALEA adaptation engine performs inference by a forward chaining mechanism. The inference starts by firing the starting rule for a particular layer. The rule-based inference used in the adaptation model is implemented as a separate module independent from the other parts of the ALEA system. The integration of this module with the other ALEA modules is realised through the particular actions in the 'then' part of the rules by using different namespaces for the particular modules. For each ALEA module there is a prefix that determines the target for an action execution.

Learning sequence layer

The learning sequence layer contains a set of rules that determine which of the two learning approaches of composing sequences of concepts ("from general to concrete" or "from concrete to general") to be currently used. The rules define heuristics for determining the appropriate learning strategy with conditions specifying when particular learning approach is useful and actions specifying numeric score change. For example, the heuristic that it is useful to start with providing information related to a program schema is expressed by the following rule:

- If a user visited less than 10% of application domain concepts, the numeric score of the "from general to concrete" approach is to be increased by 20.

The constants (10%, 20) here are determined empirically.

ALEA infers the most suitable approach for the student in the particular context. The result of rules evaluation is represented as a numeric score. The learning approach with the greater numeric score is selected as the more suitable. Re-evaluation of the suitability of a particular learning approach is invoked by the system and is primarily based on the extent of practical work done by a particular user.

The approach "from general to concrete" is more suitable for beginners, i.e. students with almost no practice. Such students are usually not prepared to solve problems individually. Even if they are able to define a function with the desired input/output behaviour, the programming style and conformance to a programming paradigm would most probably pose a problem. Rules for the learning sequence selection are specified according to this heuristic based on the data showing visited and learned concepts and the proportion of concepts, already offered and displayed to the learner (through a local guidance) and selected directly by a learner (using the context links or the contents).

Concept layer

The selection of concepts is driven by the chosen learning sequence selection approach. The concept layer is also represented by a set of rules. These rules specify the links which will be displayed, the order and type of these links, and the next concept that the user will see after he clicks on the forward link or navigates a context link. The concept selection process starts each time a user requests a concept, i.e. each time he clicks on a link, by loading the rules for the currently used learning selection sequence approach. The result of the rule set interpretation is an information structure containing a list of links related to the requested concept. However, when a learner requests guidance by clicking on the forward link, the process of link selection described above is preceded by the process of selection of the "next" concept to be displayed. The system loads a defined set of rules and the result of their interpretation is a sequence of one or more concepts that the system suggests for the learner to visit.

The concept selection process was recently enhanced by the development of an external recommender system that discovers patterns in students' behaviour during learning and provides additional list of concepts based on discovered patterns [15] (see Fig 5).

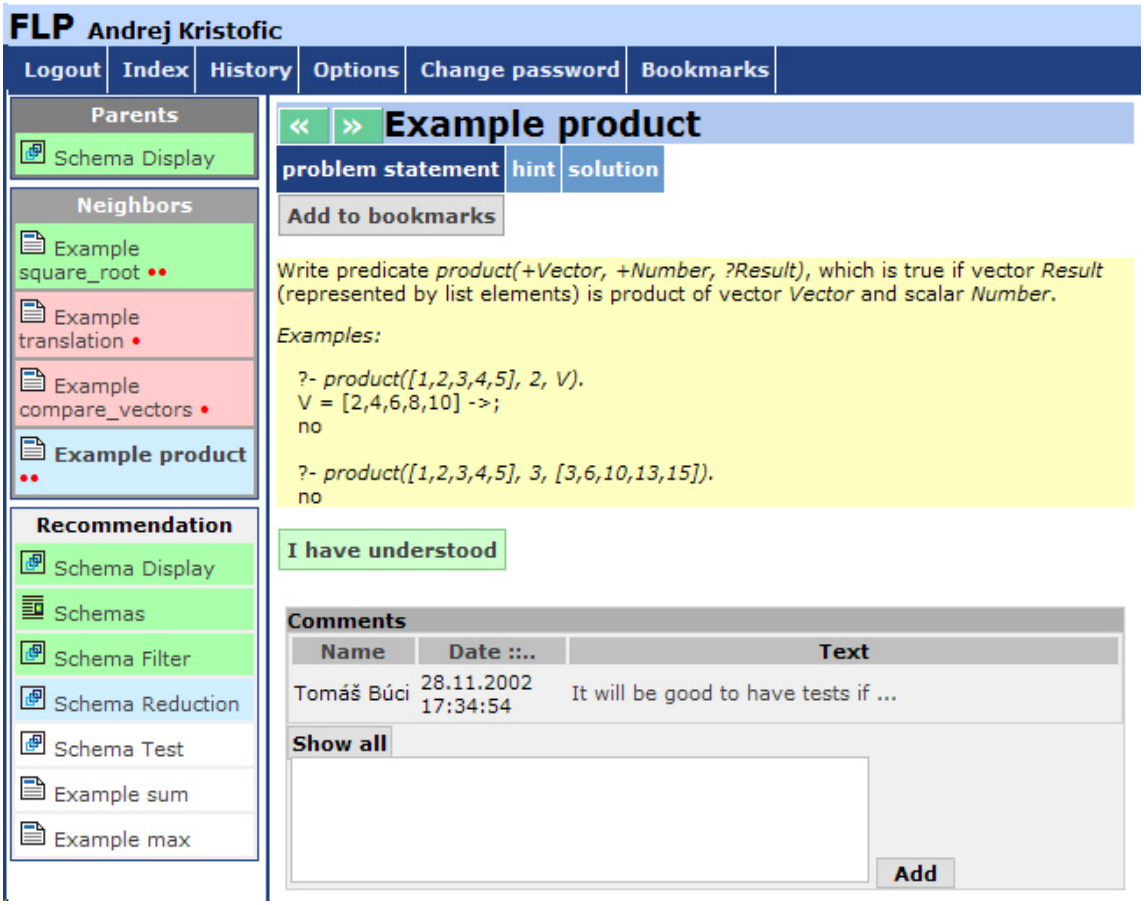

Figure 5. External recommendations incorporated into ALEA.

Information fragment layer

The aim of the information fragment layer is to select fragments from the information fragments base and produce the resulting layout of the concept presentation. The main criterion for selecting fragments for a presentation is the type of the selected concept. For example, the type of the

‘program example’ concept is normally related to the fragments of three types: definition, hints and a solution.

ALEA uses two basic alternatives for a layout of the page containing a program example. The first option is to arrange fragments using tabs (on different pages) as depicted in Fig. 3. The second option is to arrange all fragments related to the program example on one page. This option is more suitable for less experienced users. It is obvious that the solution fragment may not be displayed as the first one. A combination of the above mentioned approaches is also possible, which places the definition and the hints on one tab and the solution on the other tab.

3.2 Additional features

The main purpose of ALEA is to help novice programmers in acquiring programming skills. However, supporting teacher – student communication is very important in the teaching/learning process. So we augmented ALEA with basic communication facilities.

The first one is the opportunity to append custom comments to each displayed concept. These comments are visible to other users (see Fig. 3) and allow students to cooperate during the learning process. Instructors’ comments are displayed using a different colour from the colour used for students. The instructor has the option to remove comments. Comments may serve also as a means for assessment of students’ comprehension or their activities within the ALEA system.

Another feature is the opportunity of learning material uploading, which changes the domain model by adding new concepts and corresponding information fragments. Currently, ALEA supports only adding new program examples. Students can add a ‘program example’ concept related to any non-example concept (e.g., program schema). The only information they have to provide is a description of the programming task and its definition. Neither the hints nor the solution are required. Such an uploaded concept is displayed using different icon, to signalize that the instructor has not approved it yet. The instructor has the option to approve or remove the program example and to edit the contents of the newly uploaded concept. After a successful approval by the instructor, the program example is no longer displayed with a different icon and it is like other concepts defined in the course preparation phase.

4 Discussion

ALEA was developed as a general-purpose adaptive web-based educational system during the year of 2001/2002. We created the information content related to learning functional and logic programming in 2002, after several years of successful use of the program schemata method in our teaching of functional and logic programming. Since the academic year 2002/2003 it has been used regularly in the Functional and Logic Programming course, which is part of the undergraduate programme in Informatics at the Slovak University of Technology in Bratislava. About 100 students take this course each year.

We have evaluated our approach and ALEA focusing on two main aspects:

- To prove or refute the assumption that teaching programming with program schemata is more efficient and that the students’ progress is much faster (especially during the early stages of the learning process),
- To prove that ALEA serves well its purpose to help students in acquiring practice in programming.

With regard to the first aspect, we have started the evaluation before the design of the ALEA system. We conducted experiments containing tests for which the students were divided according to two criteria: whether they were provided with an explanation of the program schemata and whether there was a catalogue of program schemata available during the test. The conclusion was that the knowledge of the program schemata was instrumental in achieving better results. The difference between the students with prior information on program schemata and those who were provided with no explanation prior to the test supported the hypothesis about the positive influence of program schemata on learning functional programming. The students, who were given instruction related to the program schemata beforehand (even if they did not have the catalogue of the program schemata at hand during the test) performed better.

Since the ALEA system was first deployed, it has been used by nearly 300 undergraduate students in their third year of study as part of Functional and Logic Programming course. The course consists of two parts: functional programming and logic programming, half a semester each.

The ALEA system records actions performed by each student (e.g., selection of a new concept as a learning goal, click on the context link, marking the concept as learnt, forward as a requirement for local guidance). During ALEA usage in the functional programming part of the course the most frequently used action was selecting a context link (38%), i.e. clicks on the links displayed in the left part of ALEA screen as shown in Fig. 3. The second most frequently used action was a local guidance request (29%). These results were influenced by the fact that there were several students specialized in achieving particular learning goals (the students used ALEA to prepare for a midterm test on list processing). The mostly used information fragments were those containing the program schemata. Displaying solution fragments of program examples markedly outnumbered requirements for presentation of hint fragments. However, the common behaviour of the students was that they used hints for the first program examples related to a particular program schema and then proceeded just to the definition and solution parts.

ALEA was evaluated also by means of several questionnaires. The majority of the students responded positively. The most valuable feature reported by the students was the focused displaying of recommended concepts (context links). Several students reported that they have used the local guidance less frequently than the local orientation mainly due to their mistake in using the Web browser forward and back buttons instead of the ALEA forward and back local guidance links. Using the browser buttons accidentally instead of the ALEA guidance caused confusion. Students also confirmed that the approach “from general to concrete” is more suitable at the beginning of the programming practice.

5 Conclusions

This article addresses a web-based adaptive hypermedia application, ALEA, targeted on providing adaptive sequences of program examples supplemented by an explanation text. ALEA improves the student’s learning process by selecting the most appropriate program examples for learning programming concepts. Using the system in programming module can reduce the need for closed labs sessions. ALEA is not restricted to the area of functional or logic programming languages. The basic idea is to offer adapted sequences of program examples that are neither restricted by a programming paradigm, nor by a programming language.

Substantial idea related to the adaptation knowledge represented in ALEA is the definition of three separate layers in which the system behaviour can be adapted: the learning sequence layer, the concept layer and the fragment selection layer. The future development of ALEA will be focused on enhancing the adaptation capabilities and further experimenting with efficiency of adaptation techniques.

The described methodology, based on program schemata, was adopted for teaching programming before the implementation of ALEA. The approach follows the idea that by fast arrival to understanding of how to solve a rather modest number of basic classes of problems, students will be able to solve other more or less similar problems. The hypothesis that novices are able to devise correct functional or logic programs faster while using program schemata, if comparing to the "traditional" approach, was experimentally verified.

We are currently extending ALEA's domain model with programming examples related to the Java and C++ programming languages. We also plan on extending learner assessment capabilities by combination ALEA with a system for testing the learner's level of knowledge (we consider general purpose web-based systems for testing, such as SIETTE [16] together with a special purpose web-based system for testing programming knowledge, which is being currently developed).

Another interesting direction of future research is to consider different constraints defined by the student (for example time available for test preparation) to improve concept selection. In this research we will utilize data about students' behaviour collected in previous years.

Acknowledgements

The author would like to thank R. Kostelnik for his contributions to the design and implementation of the ALEA system. The results would not have been obtained without the support of R. Filkorn, M. Bockay and L. Nerad who helped in the preparation of the programming tasks in Lisp and Prolog. This work has been partially supported by the Science and Technology Assistance Agency under the contract No.~APVT-20-007104 and the Cultural and Educational Grant Agency of the Slovak Republic, grant No. KEGA 3/2069/04.

References

1 Brusilovsky, P. (1996) 'Methods and techniques of adaptive hypermedia', *User Modeling and User-Adapted Interaction*, Vol. 6, No. 2-3, pp. 87–129.

2 Brusilovsky, P. (2001) 'Adaptive hypermedia', *User Modeling and User-Adapted Interaction*, Vol. 11, No. 1-2, pp. 87–100.

3 Brusilovsky, P. (1999) 'Adaptive and intelligent technologies for web-based education', *Künstliche Intelligenz*, Special Issue on Intelligent Systems and Teleteaching, No. 4, pp. 19–25.

4 Weber, G. and Brusilovsky, P. (2001) 'ELM-ART: An adaptive versatile system for web-based instruction', *International Journal of Artificial Intelligence in Education*, Vol. 12, No. 4, pp. 351–384.

5 Kostelnik, R. and Bielikova, M. (2003) ‘Web-based environment using adapted sequences of programming exercises’, In: M. Benes (ed.), *Proc. of Information Systems Implementation and Modelling – ISIM 2003*, pp. 33–40.

6 Gegg-Harrison, T.S. (1996) ‘Extensible logic program schemata’, In: I. Gallagher (ed.), *Proc. of the 6th Int. Conf. on Logic Program Synthesis and Transformation*, Springer-Verlag, LNCS 1207, pp. 256–274.

7 Gegg-Harrison, T.S. (1999) ‘Exploiting Program Schemata to Teach Recursive Programming’, In: P. Brna, B. duBoulay, and H. Pain (eds.), *Learning to Build and Comprehend Complex Information Structures: Prolog as a Case Study*, Ablex, pp. 347–379.

8 Soloway, E. and Ehrlich, K. (1988) ‘Empirical studies of programming knowledge’, *IEEE Trans. on Software Engineering*, Vol. 10. No. 5, pp. 595–609.

9 Bielikova, M. and Navrat, P. (1998) ‘Use of program schemata in Lisp programming: an evaluation of its impact on learning’, *Informatica*, Vol. 9, No. 1, pp. 5–20.

10 Sollohub, C. (1991) ‘Programming templates: professional programmer knowledge needed by the novice’, *Computer Science Education*, No. 3, pp. 255–266.

11 Navrat, P. (1996) ‘A closer look at programming expertise: Critical survey of some methodological issues’, *Information and Software Technology*, Vol. 38, No. 1, pp. 37–46.

12 Merrill, D.M. (2000) ‘Instructional strategies and learning styles: Which takes precedence?’, In: R. Reiser and J. Dempsey (eds.), *Trends and Issues in Instructional Technology*. Prentice Hall.

13 Wu, H., Houben, G.J. and De Bra, P. (1999) ’User modeling in adaptive hypermedia applications’, *Proc. of the Interdisciplinaire Conferentie Informatiewetenschap*, pp. 10–21.

14 Cannataro, M., Cuzzocrea, A. and Pugliese, A. (2002) ‘XAHM: an adaptive hypermedia model based on XML’, In: *Proc. of the 14th Int. Conf. on Software Engineering and Knowledge Engineering*, ACM Press, pp. 627–634.

15 Kristofic, A. and Bielikova, M. (2005) ‘Improving adaptation in web-based educational hypermedia by means of knowledge discovery’, In: *Proc. of the Sixteenth ACM Conference on Hypertext and Hypermedia*, ACM Press, pp. 184–192.

16 Rios, A., Millan,E., Trella, M. Perez-de-la-Cruz, J.L. and Conejo, R. (1999) ‘Internet based evaluation system’, In: S.P. Laojie and M. Vivet (eds.), *Proc. of the 9th World Conf. of Artificial Intelligence in Education: Open Learning Environments*, IOS Press, pp. 387–394.