FILOMAT (Niš) 11 (1997), 119–134

TOWARDS INTELLIGENT TUTORING SYSTEMS USING *LESS*

Mirjana Ivanović

Abstract. A possible enhancements of specialized object-oriented language *Less* to support intelligent tutoring systems, is presented. All enhancements take form of new classes and pseudo intelligent methods which must be implemented instead of existing ones. Resulting structure and behavior of tutoring systems is comparable to separately and independently implemented tutoring systems.

1. Introduction

Information presentation as a result of a suitable information representation become very popular in the last several years. There are a lot of different systems and tools with or without intelligence, which are to be used in different educational processes. The goals of this field is (according to [10, 11, 12, 14]):

- to develop general structures, methods and tools (possibly knowledge based) for representation of information,
- to develop general methods for presentation of those information and to use them in various applications.

Now there are a lot of new trends in designing and implementing [4] educational systems, which brings a need to use and incorporate in them some intelligent behavior.

As a part of the research work at the Institute of Mathematics in the field of Computer Aided Instruction systems, a specialized language for representation and presentation of information is defined [3, 5]. The original version of language which is called *Less* is presented in [3]. Some aspects of *Less* as pure programming language are presented in [7], while other aspects of *Less* as a language suitable for multimedial representation and presentation

Received November 19, 1996; Revised October 17, 1997

1991 *Mathematics Subject Classification:* 68N20

Key words and phrases: Object-oriented methodology, Intelligent tutoring systems, Artificial intelligence

of language are presented in [5]. It is object-oriented and supports representation of multimedial information (text, picture, animation, sound) suitable for creation frameworks for developing multimedial Authoring systems [8]. *Less* has been designed as a "machine language" for one class of information (re)presentation systems, a practical tool for their implementation as well as a language for description / formalization of those systems. Besides that, *Less* can be used to produce concrete applications in implemented systems.

Less can be easily programmed to mimic existing systems such as HyperText [6] and Authoring Systems [8] as well to implement completely new systems. Although the design of *Less* was initially directed toward supporting of non-intelligent, general purpose information (re)presentation systems (like HyperText, Authoring Systems, General Presentation Packages), it offers means for implementation of intelligent systems as well. This paper discusses the possibilities to implement intelligent tutoring systems using *Less* and reveals the spots in data structure (classes) where pseudo intelligent methods could be attached. Proposed material is not limited to (specific) *Less* programming language, but also to every object oriented data structure.

2. Specialized object-oriented language *Less*

Less [5] has been designed as a tool for creation of different kinds of applications in the field of information representation and presentation. Using it, various kinds of systems like Hypertext and Authoring systems can be created. Kernel of *Less* consists of different basic data structures for information representation, and different statements and directives for information presentation. Basic classes give opportunity for storage of multumedial information. *Less* also possesses mechanisms for creating new structures.

A more classical view to *Less* would characterize it as an Authoring language that can be easily adapted to behave like Authoring system.

The representation domain (lesson knowledge) can be observed as a complex set of connected topics. The most appropriate structure for a set of topics is a multi–digraph. The graph nodes are topics which contain information and knowledge, and the graph edges are the elements of the paths through the graph during the presentation, i.e. learning.

2.1. Primitive Data Types in *Less*. As primitive data types the following can be used in all nodes of multi-digraph: `STRING`, `LOGICAL`, `NUMBER`, `TEXT`, `PICTURE`, `ANIMATION`, `SOUND` and `METHOD`.

Basic structured data types are the following:

- `LIST OF Type` - the list of arbitrary number of elements of data type `Type`.

- T(Type$_1$, ... , Type$_n$) – n-tuple of elements of arbitrary data types Type$_1$, ... , Type$_n$.
- (Val$_1$, ... , Val$_n$) – enumerated type, where Val$_1$, ... , Val$_n$ determine the set of possible values of the type.

2.2. Primitive Classes. The four primitive classes *Text*, *Picture*, *Animation* and *Sound*, are basic elements of the language. They have the same structure:

- a binding variable for representation of some kind of information (like text, picture, animation and sound),
- a method for presentation of represented information.

These four classes are embedded into class *Info*. Besides them *Info* contains the method that determines the way and sequence of information presentation, and some additional fields. Its structure can be depicted as follows:

```
Info = CLASS {
  ident: STRING; (* unique object identification *)
  key  : LIST OF STRING;(* list of keywords
                           connected to the object *)
  tex  : CLASS  Text; (* represented information *)
  pic  : CLASS  Picture;
  ani  : CLASS  Animation;
  sou  : CLASS  Sound;
  interAll : METHOD(* for information presentation *) }
```

Topic is the heart of a multi–digraph structure. It is intended to store information of two kinds: information to be presented and information based on which a decision on further movement direction can be made. *Topic* is made of class *Info* that contains information to be presented and different classes that contains tasks to be posed after *Info* is presented.

All tasks to be solved have the common core *LTask*.

The more concrete types of tasks (multiple choices, typed-in answers, etc.) are represented with different user defined classes, which embeded class *LTask*.

Finally, the class *Action* holds an identifier of a topic where presentation will continue. By choosing different strategies for placing the contents into objects of *Action* class, different information presentation systems can be implemented.

2.3. Basic parts of *Less* program. *Less* program can consist of three parts

1) Part for user-defined structures, where new classes and structure of whole system will be defined.
2) Part of user-defined methods, where user can redefine standard public methods which participate in process of presentation, and can define new, very specific, private methods which can be treated as auxiliary methods.
3) Part of setting a multi-digraph, where every topic is filled-in with appropriate contents. This part is equivalent to phase of information representation in system. In this part user can use different kinds of directives and statements.

3. Authoring systems and intelligent tutoring systems

Authoring systems (also known as Computer Aided Instruction – CAI systems) can be designated as non–intelligent, general purpose systems with feedback. They are applied in computer aided instruction and general presentations. Often they integrate several peripherals in so–called multi–medial environment.

Lesson to be presented is divided into smaller pieces called topics which are somehow connected in semantic entirety. Topics and links between them form a multi–directed graph according to a prescribed lesson plan. It is often useful that Authoring systems possess possibility to test the knowledge student has accepted during the process of learning. This can be achieved by posing the problems (tasks) to be solved at some check–points in = the graph. Problems can be in the form of multiple–choice tests, = relationship tests, typed–in answers to different questions, etc.

User responses to posed tasks constitute the feedback which is used by Authoring system to direct further movement through multi–digraph. This leaves Authoring systems with only one flaw – it is still hard to break the initial problem into smaller parts and connect them into suitable semantic entirety. Moreover, it is also hard to present data on different media in a synchronized manner.

Intelligent tutoring systems (ITS) are exclusively intended for learning processes. They represent a combination of different fields: education, psychology, artificial intelligence, cognitive sciences etc. These are usually highly specialized systems which is the reason why they successfully employ the methods of artificial intelligence. There are, for example, a lot of intelligent tutoring systems, devoted to very specific domains:

- Advanced Cardiac Life Support (ACLS) tutor – in which a student takes the role of team leader in providing emergency life support for patients who have had heart attacs, or CARDIAC [14] for teaching about cardiac resuscitation,

- PEG (Pedagogical Explanation of electrical circuits),
- Coach – teches how the use UNIX mail,
- MDF – is some kind of mathematics tutor.

Since they are basically intended for learning processes, ITS are systems with feedback. Their architecture and structure varies, but usually four elements can be recognized [2, 13]:

1. **Expert (teaching) module** is a data base which stores information from the field in which ITS is used.
2. **Student module** is intended for modelling individual student's knowledge in the field where ITS is used. The content of the module constantly changes during the process of learning.
3. **Tutoring module** specifies the method of presenting the material in the field in which ITS is used, the manner and rhythm of presentation.
4. **Diagnostic module** which constantly modifies the student module in accordance with answers that student gives to the posed questions. This module is also linked to the expert module.

Efforts that are made concerning further development of these systems have as their aim the communication between students and computer in natural language. However, taking into account current state of affairs in research in natural language interfaces, nowadays the communication is performed either in meta-language which is close to a natural language or in some subset of a natural language.

The communication is established by translating a natural language into a computer code at the input of the system, while at the output machine code is translated into a 'natural' language. The communication is mainly textual or has some rudimentary graphics capabilities.

4. From authoring systems to intelligent tutoring systems in *Less*

As intelligent tutoring systems are highly specialized, a large team (programmers, domain experts, education theorists,...) is needed to create one ITS. Also, one of the main difficulties in making ITS is the time and cost required. So called, Authoring tools try to obtain mechanisms to reach easier concrete ITS. They try to provide simple development environment and as a consequence fewer number of developers would be needed for the construction of educational software.

To achieve this goal there are two main approaches [2]:

1) to provide development shell for educators to prepare their own courses,
2) to provide a means for programers to represent, more easily, the domain and theaching strategies.

In this paper one version of Authoring tool is presented. LAT (*Less* Authoring Tool) is an authoring tool based on programming language *Less*. LAT offers planty of different building blocks (classes) and means of their interconnection into Authoring system. Teachers (without computer experience) can use them to preare lessons in different domains. After that, together with a programmer, they, in routined and easy way, can complete lesson and store it in computer i.e. make it ready for the students.Using that approach, available blocks, mechanisms (in *Less*) for building new blocks and some methods to maintain students model, it is possible to create some intelligent tutoring systems. In this section some basic elements for building Authoring systems are given.

Lesson that should be represented in Authoring system is divided into several topics. Every topic contains tasks which have to be solved during the presentation of information, i.e. learning. The movements through a multidigraph depend on previous specific and general knowledge of the student. For that reason it is useful to record all paths through the graph made by students and use that information in determining the success and quality of the lesson and learning process. A possible structure for that purpose can be class *Student*. Suggested structure is rather simple because is not aware of previous student's knowledge. It is however suitable for storing different statistic data.

```
Student = CLASS {
  ident   :   STRING;
  success :   LIST OF CLASS  STopic }
```

where

- **ident** – is unique identification of the student.
- **success** – is a list of objects where identifiers of topics and tasks are stored, and contains other information about learning process and its success.

STopic is a new class for that purpose. It has the following structure:

```
STopic = CLASS {
  idTopic :  STRING;
  solAns  :  LIST OF T(NUMBER,LOGICAL) }
```

where

- **idTopic** is unique identifier of object, i.e. topic the student has passed during learning.

- **solAns** is a list of pairs. Every element is a pair whose first component stores a number of task for the topic. The second component is logical value showing whether the task is successfully completed during the current process of learning.

Class *LTopic* is introduced to enable a representation of topics of a lesson.

```
LTopic = CLASS {
  ident : STRING;
  CLASS  Info EXCL ident;
  query : LIST OF CLASS
          ( LTaskP,  LTaskT,  LTaskR,  LTaskB, =
LTaskE) }
```

LTopic inherits the class *Info* without the variable ident. Besides that, it contains the following fields more:

- **ident** is unique object identifier.
- **query** is list of objects of the classes *LTaskP*, *LTaskT*, *LTaskR*, *LTaskB* or *LTaskE*. The object represents queries, tasks or exercises which are attached to the topic. Student should pass through all tasks, after the presentation of all information from the topic. Depending on student's solutions to posed tasks, further presentation is directed in different parts of the graph.

LTask is common structure for all kinds of tasks and has the following form:

```
LTask = CLASS {
  ident    :  STRING;
  CLASS  Info EXCL ident;
  aldSlvd :  LOGICAL;
  disp     :  LOGICAL;
  dispLT   :  PUBLIC METHOD}
```

Class *LTask* inherits the class *Info* for representation of information on the task to be solved. Besides that *LTask* contains several fields more:

- **ident** is unique identifier of the task.
- **aldSlvd** contains logical truth value **TRUE**, if the tasks has been solved previously. Otherwise it contains the value **FALSE**. This field can be used to skip over already solved tasks during repeated passes through the topic.
- **disp** determines if the correct answers will be displayed after the student has given a wrong one.

- **dispLT** is public method which arranges all information on the screen and accept student's response. All instances of a *LTask* class use the same method, i.e. the same arrangement of information on the screen.

The mentioned fields are fixed in every task. In the following, we shortly describe some characteristic task types.

- *LTaskP* expects longer student's answers which should be explicitly typed–in. This class has following structure:

```
LTaskP = CLASS {
  CLASS  LTask;
  correct :  STRING;
  answer  :  STRING;
  isCorr  :  PRIVATE METHOD;
  ifCorr  :  CLASS  Action;
  ifNot   :  CLASS  Action }
```

- *LTaskT* expects that the student will choose one of several possible alternatives. This class has following structure:

```
LTaskT = CLASS {
  CLASS  LTask;
  answer :  LIST OF CLASS  Alter }
```

- *LTaskR* expects that the student will determine correct pairs out of two independent lists of components. This class has following structure:

```
LTaskR = CLASS {
  CLASS  LTask;
  answer1 :  LIST OF TEXT;
  answer2 :  LIST OF TEXT;
  true    :  LIST OF NUMBER;
  isCorr  :  PRIVATE METHOD;
  ifCorr  :  CLASS  Action;
  ifNot   :  CLASS  Action }
```

- *LTaskB* expects simple yes or no answer. This class has following structure:

```
LTaskB = CLASS {
  CLASS  LTask;
  correct :  LOGICAL;
  answer  :  LOGICAL;
```

```
    isCorr  :  PRIVATE METHOD;
    ifCorr  :  CLASS  Action;
    ifNot   :  CLASS  Action }
```

- *LTaskE* is longer procedure or exercise which the student should execute. This class has following structure:

```
LTaskE = CLASS {
  CLASS  LTask;
  exeFil :  STRING;
  next   :  CLASS  Action }
```

Multiple–choice answers (of type LTaskT) can be represented using the following class:

```
Alter = CLASS {
  ident  :  STRING;
  CLASS   Info EXCL ident, key, sou;
  act    :  CLASS  Action;
  isCorr :  LOGICAL }
```

where

- act contains action which should be taken if the student choose that alternative as a solution to posed task.
- isCorr contains the value **TRUE** if the alternative is correct one.

Finally, the *Action* class describes the actions which should be taken if a particular alternative has been chosen. Possible actions are: the end of presentation; a jump to another topic in the graph; and a jump to the other task in the same topic.

Action can be defined in the following way:

```
Action = CLASS {
  en  : END;
  Top : IDENT  LTopic;
  Tsk : IDENT
        ( LTaskP,  LTaskT,  LTaskR,  LTaskB,  =
LTaskE) }
```

where the value en designates the end of presentation process. The value Top designates the jump to another topic in the graph. In that case an unique identifier of the target topic is attached. The value Tsk designates the jump to another task in the same topic and in that case an unique identifier of the target task is attached.

By using all described classes, a multi–digraph of a whole lesson can be represented as the class *Lesson*, in the following way:

```
Lesson = CLASS {
 ident   : STRING;
 lsn     : LIST OF CLASS  LTopic;
 start   : LIST OF IDENT  LTopic;
 stop    : LIST OF IDENT  LTopic;
 currTpc : IDENT  LTopic;
 currTsk : IDENT  LTask;
 interpr : PUBLIC METHOD;
 stud    : LIST OF  Student }
```

where

- **ident** is a short name of a lesson (multi–digraph).
- **lsn** is a list of topics which cover content of lesson. Topics are connected and form a graph.
- **start** and **stop** are the lists of identifiers of starting (i.e. ending) topics of the lesson.
- **currTpc** and **currTsk** always points to the current topic and current task in the topic.
- **interpr** is a method which supports the learning process.
- **stud** is a list of objects for every student who use the lesson. It can consists of various statistic data and other relevant information about students.

Just described classes constitute a definition of a general Authoring system and it is the first part of the *Less* program. More specific methods which support concrete presentation can be defined in the second part of the *Less* program. The last part of the *Less* program contains the content of the structure which is defined in the first two parts. When such *Less* program is executed, an multi-digraph is formed and filled-in, and Authoring system is ready for use, i.e. for the start of a presentation.

Suggested data structure for a representation of a lesson is a good basis for development of (pseudo)intelligent tutoring systems. In the suggested Authoring system:

1) **Expert module** is realized by list of *LTopic*'s objects.
2) **Student** and **Diagnostic** modules are weak point of the suggested system. But, they can be realized using capabilities which *Less* offers, in relatively rudimentary, but satisfactory way.
3) **Tutoring module** is supported by defined multidigraph i.e. by making connections between topics and tasks in determining paths

through topics. It supports the teacher's own way of presentation lesson's material.

Intelligent functions can be embedded into the described approach in several ways:

- Automated lesson creation.
- Intelligent communication between student and lesson.
- Extensions to the student's part of a lesson.

First and second enhancement coused great efforts and require big team of different experst, and evolve development of general artificial intelligent techniques and methods. Also we try to propose general tool (independant of teaching domain) with some intelligent functions. Much easier, to include some intelligence in our authoring tool is to add new classes and methods to support qualitative connection betwean models in ITS.

Automated lesson creation

Lesson could be automatically created on the basis of a whole lesson text and all the figures, animations and so on. The author of the lesson should in that case associate to certain parts the lists of keywords. Based on keywords, an intelligent program could divide the whole text into logical entireties (topics) and even offer the most appropriate paths through the lesson.

Based on the same principle, tasks for every topic could be created and attached. While for the creation of topics, only classical methods of data flow analyses could be sufficient, creation of tasks based on a plain text should involve elements of natural language understanding [1, 9], as well as substantial knowledge on described topics.

However, the lesson will take its final form by the human lesson creator.

The implementation of automated task creation is too advanced for the current state of the art in the field of artificial intelligence, except perhaps for the more exact and more specific topics in mathematics, physics and similar fields.

Intelligent Communication

Specific intelligent behavior can be attached to every task by means of private methods. One of the most appropriate spots where intelligent methods can be used are the tasks of the type **Prompt**. In its most rudimentary version, it is expected that the student's answer literally match preloaded correct answer. On the other hand, natural language understanding [1, 9] could be involved and in that case other types of tasks will practically be not needed.

Between these two possibilities, there is a whole range of possibilities for matching of student's answers to correct one. Methods could be purely syntactical ones and involve algorithms for ignoring the typing errors; or

could also involve some minor knowledge on the task posed and be aware of some synonyms.

Extensions of the student's part of a lesson

Some classes presented in privious text can be modified to include "intelligent" functions in authoring tool to obtain pseudo intelligent tutoring system.

Class *Student* can be modified to support functions of **Student's module** in ITS. The student model stores information that is specific to each individual learner. At a minimum such a model tracks how well a student is performing on the material being thought. According to that original class Student fulfills minimal criteria but, for example, class Student can be enriched by **indi_know** field and **diagnostic** method:

```
Student = CLASS {
  ident    :   STRING;
   indi_know:      LIST OF  LTopic;
  success  :   LIST OF  STopic;
   diagnostic:     METHOD  }
```

where

- **indi_know** – is list of topics where individual student's knowledge is stored.
- **diagnostic** is method which constantly modifies class *STopic* according to student's knowledge and answers to the posed tasks during the process of learning.

To do that class *STopic* also has to be modified:

```
STopic = CLASS {
  idTopic  :  STRING;
  solAns   :  LIST OF
              T(NUMBER,LOGICAL,  LIST OF IDENT  LTopic )}
```

where the third component of **solAns** is a list of topics which student may visit (if wonts) to complete and improve his/her knowledge. Method **diagnostic** uses available additional topics and lessons topics to determine topics which help student to improve his/her knowledge. It has to possess some intelligent functions to discover, among available topics, relevant and connected to the question student failed. The realization of the method is still under consideration, but general intention is to incorporate some artificial intelligence techniques and algorithms.

Classes *LTask* and *Lesson* can be also modified to enriche some desirable functions.

```
LTask = CLASS {
  ident        : STRING;
  CLASS  Info EXCL ident;
   additional : LIST OF IDENT  Info;
  aldSlvd      : LOGICAL;
  disp         : LOGICAL;
  dispLT       : PUBLIC METHOD}
```

where **additional** is list of identifications of additional topics available for that task.

```
Lesson = CLASS {
 ident    : STRING;
 lsn      : LIST OF CLASS  LTopic;
  gen      : LIST OF CLASS  LTopic;
 start    : LIST OF IDENT  LTopic;
 stop     : LIST OF IDENT  LTopic;
 currTpc  : IDENT  LTopic;
 currTsk  : IDENT  LTask;
 interpr  : PUBLIC METHOD;
 stud     :          LIST OF  Student }
```

where **gen** is a list of topics which contain some notion, information or knowledge as additional material to the lesson.

Method **interpr** an be extended so that at the beginning of the process of the presentation it can ask student to enter manner of presentation. On that way, the tutoring module can be supported.

In the rest of that section global structure of some methods is presented.

```
PROCEDURE interpr;
BEGIN
    (* display Ident of all starting methods *)
    WHILE NOT EndOfList(start) DO
         Write(start.ident); start:= Next(start)
    END;
    (* user choose a topic as the starting one *)
    Read(currTpc);
    chooseManerOfLearning;
    (* presentation starts from the chosen topic
       in specified maner *)
    WHILE currTpc <> NIL DO
```

```
        currTpc.interAll;
        (* displays information from the topic *)
        currTsk := currTpc.query; (* take first task *)
        WHILE currTsk <> NIL DO currTsk.dispT END
        (* displays first task and accepts answer *)
    END
END interpr.


PROCEDURE dispT;
...
BEGIN
    interAll; (* explaines the problem *)
    CASE type OF
    prompt:
        Read(answer);
        IF isCorr THEN (* if answer is correct *)
            ProceedWith(ifCorr)
            (* takes further action *)
        ELSE
            IF disp THEN
                (* displays correct answer *)
            END;
            Proceed With(ifNot)
        END;
    ....
    (* processes other types of tasks *)
    END;
    diagnostic(Student, Topic);
    END;
END dispT.

PROCEDURE diagnostic(Student, Topic);
BEGIN
    IF NOT exist(Student, task) THEN
        newObject(Student.success)
    END;
    changeInformationAboutTask;
    IF additionalKnowledge is needed THEN
        presentAdditionalKnowledge
    END
```

```
END diagnostic.
```

5. Conclusion and related work

Less is specialized object oriented language designed for building of educational computer aided instruction (CAI) systems. It is relatively small and easy maintainable language and even professors without any computer knowledge can quickly learn and use it, to build their own CAI systems. It poseses constructs and different building blocks which highly follow procedure for lesson preparation in our elementary and high schools and teachers can easily adapt their materials to computer presentation. Based on *Less*, an authoring tool LAT is designed which possesses some pseudo intelligent functions which can help teachers to improve their lessons. Why *Less* and LAT? Our elementary and high schools usually do not have appropriate hardware to support authoring and intelligent tutoring systems available on the world's markets. Appart from that they are ussually expensive. On the other hand, LAT possesses good performances and can run on poorer hardware configurations, it is easier for use and free of charge.

Available structures in LAT easily follow standard process of preparing and making lessons, so only one step is needed to transfer them in computer and use computer in educational processes. In that way professor only with help of one programer can create his/her shell for different lessons.

Once create CAI systems' shell professor can quickly and simple prepare different lessons in multimedial manner. They need not either powerful hardware nor software to achieve satisfactory multimedial presentations.

By implementing appropriate methods in different parts of a *Less* program, it is possible to use *Less* to build intelligent tutoring systems. Even if professor is skilled programer he/she can (but, it is not so easy) replace standard methods (available in *Less*) by his/her own. Knowing that in the heart of a *Less* design is an object oriented data structure, the paper presents also the way to build intelligent information (re)presentation systems on top of an object-oriented layer[1].

Refrences

[1] J. Allen, *Natural Language Understanding*, The Benjamin/Cummings Publishing Co., Inc., 1987.

[2] J. Beck, M. Stern, E. Haugsjaa, *Applications of AI in Education*, Crossroads, Fall 1996., pp. 11–15.

[1]This work belongs to the field of informatics (artificial intelligence and expert systems) and has strong (although not direct) impetus to all the other fileds.

[3] Z. Budimac, M. Ivanović, *On Specialized Language for Information (Re)-presentation*, Proc. DECSYM '92 Conference (Side-Antalia, Turkey), 1992, pp. 175–187.

[4] Communications of the ACM, *Learner-Centered Design*, 39(1996)4.

[5] M. Ivanović, *Contribution to the Development of Programming Languages using Object-Oriented Methodology*, Ph.D. Thesis, University of Novi Sad, (in Serbian), 1992.

[6] M. Ivanović, *A HyperText Shell Using Less*, Novi Sad Journal of Mathematics, 26(1996)1, pp. 135–153.

[7] M. Ivanović, Z. Budimac, *Less - An Object Oriented Programming Language*, Proc. Int. Summer School with Conference "Information Technologies and Programming" (Sofia, Bulgaria), 1992, pp. 77–82.

[8] M. Ivanović, Z. Budimac, *A Framework for Developing Multimedial Authoring Systems*, Proc. of 1996 IEEE International Conference on Systems, Man and Cybernetics, Beijing, China, 1996, pp. 1333–1338

[9] H. Kamp, U. Reyle, *From Discourse to Logic. Introduction to Model Theoretic Semantics of Natural Language, Formal Logic and Discourse Representation theory*, Studies in Linguistics and Philosophy, Kluwer Academic Publisher, 1993.

[10] A.A. Rodriguez, L.A. Rowe, *Multimedia Systems and Applications*, IEEE Computer, May 1995, pp. 20–22.

[11] R. Sharon, R. Bell, *Tools that bind: Creating Integrated Environments*, IEEE Software, March 1995, pp. 76–85.

[12] W. Wahlster, *Knowledge Based Information Presentation*, Notes from the Lecture Given at Summer School of AI, Dubrovnik, September 1990.

[13] P.H. Wood, *Intelligent Tutoring Systems: An Annotated Bibliography*, SIGART Bulletin, 1(1990), pp. 21–41.

[14] B.P. Woolf, W. Hall, *Multimedial pedagogues, Interactive Systems for Teaching and Learning*, IEEE Computer, May 1995, pp. 74–80.

University of Novi Sad, Faculty of Science, Institute of Mathematics,
Trg D. Obradovića 4, 21000 Novi Sad, Yugoslavia
E-mail: mira@unsim.ns.ac.yu