ELSEVIER

Expert Systems with Applications 26 (2004) 477–492

Expert Systems with Applications

www.elsevier.com/locate/eswa

Using a hybrid rule-based approach in developing an intelligent tutoring system with knowledge acquisition and update capabilities

Ioannis Hatzilygeroudis[a,b,*], Jim Prentzas[a,b]

[a]*Department of Computer Engineering and Informatics, School of Engineering, University of Patras, 26500 Patras, Hellas, Greece*
[b]*Research Academic Computer Technology Institute, P.O. Box 1122, 26110 Patras, Hellas, Greece*

Received 14 September 2003; revised 14 September 2003; accepted 28 October 2003

Abstract

In this paper, we present the architecture and describe the functionality of an Intelligent Tutoring System (ITS), which uses an expert system to make decisions during the teaching process. The expert system uses neurules for knowledge representation of the pedagogical knowledge. Neurules are a type of hybrid rules integrating symbolic rules with neurocomputing. The expert system consists of three components: the user modelling unit, the pedagogical unit and the inference system. The pedagogical knowledge is distributed in a number of neurule bases within the user modelling and the pedagogical unit. Another important component of the ITS, for both its development and maintenance, is its knowledge management unit, which provides knowledge acquisition and knowledge update capabilities to the system, that is, offers expert knowledge authoring capabilities to the system.

Keywords: Hybrid rule-based systems; Knowledge acquisition; Knowledge revision; Intelligent tutoring systems; Neurocomputing

1. Introduction

Intelligent Tutoring Systems (ITSs) form an advanced generation of Computer Aided Instruction (CAI) systems Their key feature is their ability to provide a user-adapted presentation of the teaching material (Polson & Richardson, 1988). This is accomplished by using Artificial Intelligence (AI) methods to represent the pedagogical decisions and the information regarding each student (Angelides & Garcia, 1993; El-Khouly, Far, & Koono, 2000; Georgouli, 2002; Vassileva, 1997). The emergence of the World Wide Web increased the usefulness of such systems (Brusilovsky, Schwarz, & Weber, 1996; Hwang, 1998; Moundridou & Virvou, 2003; Stern & Woolf, 1998).

In any case, the AI techniques that an ITS employs is a very significant issue for its development, operation and maintenance. The gradual advances in AI methods have been incorporated into ITSs resulting into more effective systems (Urretavizscaya-Loinaz & Fernandez de Castro, 2002). During the past years, various AI formalisms have been developed for knowledge representation in knowledge-based systems, such as symbolic rules, conceptual graphs, fuzzy logic, Bayesian networks, neural networks, case-based reasoning, etc. Most of them have been used for knowledge representation in ITSs (Hwang, 2003; Josephina & Nkambou, 2002; Nkambou, 1997; Shiri, Aimeur, & Frassen, 1998; Zhendong, 2001). Symbolic rules are perhaps the most prominent AI formalism used in ITSs (Simic & Devedzic, 2003; Vassileva, 1997). Hybrid knowledge representation approaches, integrating two or more formalisms (e.g. symbolic–symbolic, neuro-symbolic or neuro-fuzzy representations), have also been developed in an effort to create improved representations (Medsker, 1995; Nauck, Klawonn, & Kruse, 1997; Sun & Alexandre, 1997). Till now, a few ITSs are based on hybrid formalisms (Magoulas, Papanikolaou, & Grigoriadou, 2001; Prentzas, Hatzilygeroudis, & Garofalakis, 2002). However, hybrid approaches can offer a number of benefits to ITSs not offered by single ones.

In this paper, we present the architecture and describe the functionality of a Web-based ITS. The ITS uses an expert system to make decisions during the teaching process. Expert knowledge is represented via a type of hybrid rules,

* Corresponding author. Address: Department of Computer Engineering and Informatics, School of Engineering, University of Patras, 26500 Patras, Hellas, Greece. Tel.: +30-2610-996937; fax: +30-2610-960374.
E-mail addresses: ihatz@ceid.upatras.gr, ihatz@cti.gr (I. Hatzilygeroudis); prentzas@ceid.upatras.gr (J. Prentzas).

0957-4174/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2003.10.007

which offer a number of benefits, such as time and space efficiency and reasoning robustness. The ITS also includes a knowledge management unit (KMU), which provides semi-automated knowledge acquisition and knowledge update capabilities.

The paper is organized as follows. Section 2 presents an overview of the system's architecture. Section 3 presents the hybrid knowledge representation formalism. Section 4 presents features of the domain knowledge. In Section 5, the expert system and its components are described. Section 6 presents the functionality of the system supervisor, the component that controls the whole tutoring process. In Section 7, the knowledge management component is presented. In Section 8, the benefits of using the hybrid rules are outlined. Finally, Section 9 concludes.

2. System overview

Fig. 1 depicts the basic architecture of the ITS. It consists of the following components: system supervisor, knowledge management unit, domain knowledge, user modelling unit, pedagogical unit and inference system.

The last three components constitute the core of a *hybrid expert system*, which is used for decision making during the teaching process. The expert system employs a hybrid knowledge representation formalism, called *neurules* (Hatzilygeroudis and Prentzas, 2000). Neurules look like symbolic (if–then) rules, although more complex. The neurules of the system are distributed, according to their functionality, into different neurule bases. More specifically, there are five neurule bases, two in the user modelling unit and three in the pedagogical unit. The *user modelling unit* deals with the construction and update of the user (student) models. The *pedagogical unit* is used to adapt the lesson plan to the current user model, by making decisions on the teaching strategy, the teaching concepts and the teaching material. The *inference system* applies to any of the knowledge (neurule) bases to produce corresponding conclusions.

The *system supervisor* controls the overall function of the ITS. It interacts with the other components of the ITS calling the inference system whenever it is necessary. Furthermore, it plays a user interface role.

The teaching subject of the ITS is 'Internet Technologies', including topics like: 'Basic aspects of computer networks', 'Internet and its basic services', 'WWW', etc. So, the *domain knowledge* component includes information and material related to the teaching subject.

Finally, the *knowledge management unit* is concerned with acquisition and update of the knowledge contained in the various knowledge bases of the ITS.

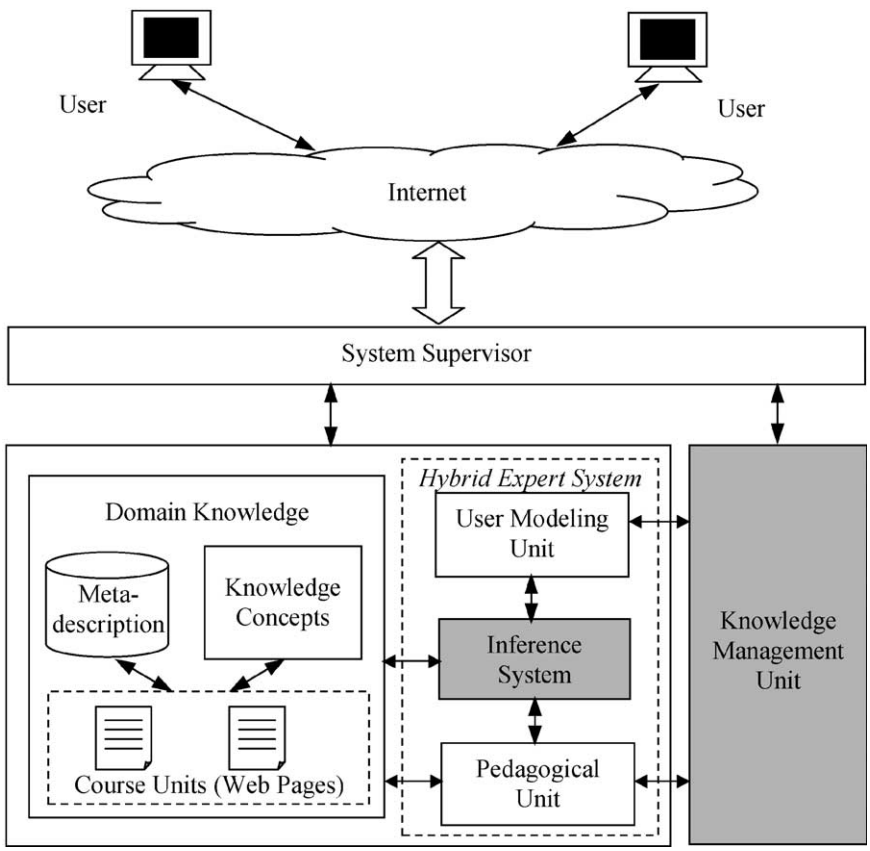

Fig. 1. The architecture of the ITS.

3. Knowledge representation

Symbolic rules constitute a popular knowledge representation scheme used in the development of knowledge-based systems (Gonzalez & Dankel, 1993). Rules exhibit a number of attractive features such as naturalness, modularity and ease of explanation. They are a typical approach of *symbolic knowledge representation*. One of their major drawbacks is that the interaction with the expert may turn out to be a bottleneck, causing delays in the system's overall development. Another drawback is the inability to draw conclusions when the value of one or more conditions is unknown.

During the last years, artificial neural networks have been quite often used in the development of expert systems. Neural networks represent a totally different approach to the problem of knowledge representation known as *connectionism* (Gallant, 1993). Some advantages of neural networks are their ability to obtain knowledge from empirical data (reducing the interaction with experts), their high level of efficiency, their ability to reach conclusions based on partially known inputs and their ability to represent complex and imprecise knowledge. A disadvantage of neural networks is the fact that they lack the naturalness and modularity of symbolic rules. The knowledge encompassed in neural networks is in most cases incomprehensible. So, neural networks are more or less black boxes. Also, incremental development of a neural base is not possible. Finally, update of a neural base normally requires retraining of the whole network.

In our system, we use neurules, a type of hybrid rules integrating symbolic rules with neurocomputing, for knowledge representation (Hatzilygeroudis & Prentzas, 2000). The form of a neurule is depicted in Fig. 2a. Each condition C_i is assigned a number sf_i, called its *significance factor*. Moreover, each rule itself is assigned a number sf_0, called its *bias factor*. Internally, each neurule is considered as an adaline unit (Fig. 2b). The *inputs* C_i ($i = 1, \ldots, n$) of the unit are the *conditions* of the rule. The weights of the unit are the significance factors of the neurule and its bias is the bias factor of the neurule. Each input takes a value from the following set of discrete values: [1 (true), 0 (false), 0.5 (unknown)]. The *output D*, which represents the *conclusion* (decision) of the rule, is calculated via the formulas

$$D = f(a), \quad a = \mathrm{sf}_0 + \sum_{i=1}^{n} \mathrm{sf}_i C_i$$

where a is the activation value and $f(x)$ the activation function, which is a threshold function

$$f(a) = \begin{cases} 1 & \text{if } a \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Hence, the output can take one of two values, '-1' and '1', representing failure and success of the rule, respectively.

The general syntax of a condition C_i and the conclusion D is

$$< \text{condition} > \ :: = < \text{variable} > < \text{l-predicate} > < \text{value} >$$
$$< \text{conclusion} > \ :: = < \text{variable} > < \text{r-predicate} > < \text{value} >$$

where $<$variable$>$ denotes a *variable* that is a symbol representing a concept in the domain, e.g. 'teaching-method', 'mark-level', etc. $<$l-predicate$>$ denotes a symbolic or a numeric predicate. The *symbolic predicates* are {is, isnot}, whereas the *numeric predicates* are $\{<, >, =\}$. $<$r-predicate$>$ can only be a symbolic predicate. $<$value$>$ denotes a value. It can be a symbol, a number or a list (see Section 5 for example neurules).

A variable in a condition can be either an *input variable* or an *intermediate variable*, whereas a variable in a conclusion can be either an intermediate or an *output variable* or both. An input variable takes values from the user (input data), whereas intermediate and output variables take values through inference, since they represent intermediate and final conclusions, respectively.

Neurules can be produced either from symbolic rules (Hatzilygeroudis & Prentzas, 2000) or from empirical data (Hatzilygeroudis & Prentzas, 2001a).

Apart from neurules, *facts* are used to represent knowledge. A fact has the form of a condition of a neurule and represents a variable-value pair.

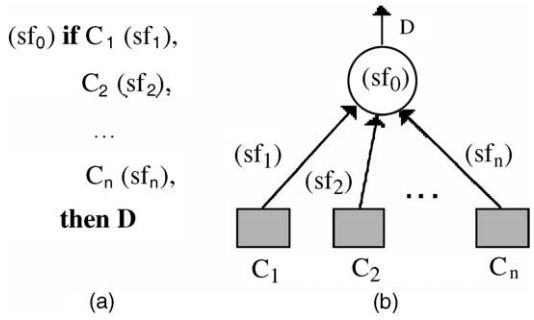

Fig. 2. (a) Form of a neurule (b) corresponding adaline unit.

4. Domain knowledge

Domain knowledge contains knowledge related to the teaching subject (domain) as well as the actual teaching material. It consists of three parts: (a) *knowledge concepts*, (b) *course units* and (c) *meta-description*.

Knowledge concepts are the elementary pieces of knowledge of the specific domain. Every concept has a number of *attributes*, such as name, difficulty-level, detail-level, acceptable-knowledge-level. Knowledge concepts are organized into *concept groups*. A concept group contains

Table 1
An example concept group with its concept subgroups and concepts

Concept group	Concept subgroups	Concepts
World Wide Web	Web structure	HTML, hypertext, Web page, Web site, Web browser
	Web access	HTTP, Web server, IP address, URL, Web browser
	Web communication	Discussion forums, usergroups

closely related concepts based on the knowledge they refer to. Therefore, the domain of the subject is dissected into sub-domains, such as 'Computer Networks' and 'World Wide Web', which are represented as concept groups. Concept groups may contain a number of concept subgroups. 'World Wide Web', for instance, contains as subgroups the 'Web structure', 'Web access' and 'Web communication'. Each concept subgroup contains a number of concepts (see Table 1).

Furthermore, each concept can have 'prerequisite to' *relationships* with other concepts. These relationships denote its prerequisite concepts. For example, concept 'HTML' may have 'prerequisite to' relationships with concepts 'Hypertext' and 'Web page'. These relationships can be viewed as links, so that a number of *concept networks* are formed representing the pedagogical structure of the domain.

Course units constitute the teaching material to be presented to the users (students) in the form of Web pages. The teaching material includes a variety of topics, ranging from introductory to advance. Each course unit is associated with a knowledge concept. The user is required to have an acceptable level of knowledge of the concept's prerequisite concepts, to be able to start a learning session concerning the knowledge contained in the corresponding course unit. In order to achieve a user-adapted presentation of the teaching material, the system keeps variants of the same page (course unit) with different presentations.

Domain knowledge also includes a *meta-description* of the course units. The meta-description concerns the *attributes* that describe the course units. The main attributes of a course unit represent its level of difficulty, its pedagogical type (theory, example, exercise), its multimedia type (e.g. text, image, animation, interactive simulation), its detail level, etc. The meta-description of the course units is based on a part of the ARIADNE metadata recommendation (http://www.ariadne-eu.org) and is stored in a relational database.

The separate representation of the domain's pedagogical structure (concept networks and units meta-description) from the actual teaching content (course units) not only facilitates updates in the domain knowledge, but also pedagogical decisions (see also Vassileva, 1997).

5. The hybrid expert system

5.1. User modelling unit

The user modelling unit is used to record user-related information, which is vital for the system's user-adapted operation. It contains models of the users of the system and mechanisms for creating and updating these models (Fig. 3). A *user model* consists of four types of items: (i) personal data, (ii) interaction parameters, (iii) student characteristics and (iv) concept knowledge.

Personal data concerns information necessary for the creation and management of the user's account, like the user's name and email. It is used for the identification of the user.

The *interaction parameters* represent information recorded during the interaction of the user with the system. They represent things like, the type and number of course units that have been accessed, the concepts and concept groups the accessed units belong to, the type and amount of assistance asked, the correct and wrong answers to tests.

Student characteristics refer to the learning abilities and the preferences of the user. There are a number of student characteristics, such as multimedia type preference, user type, concentration level, computer experience. Based on the way their values are acquired, the student characteristics are distinguished in *askable* and *inferable*. The askable characteristics, such as the multimedia type preference, are given values directly by the user, whereas the values of the inferable ones, such as the concentration level, are inferred by the system, based on the interaction parameters and the concept knowledge. So, a new user must give values to the askable student characteristics. However, the user has the option to change them during a teaching process.

Initialisation of student characteristics is made via *stereotypes* (Rich, 1989). The stereotypes represent classes of typical users. So, each user is assigned one of some predefined classes (stereotypes). The *classification module* contains a neurule base, called the *classification base*, which is responsible for selecting the appropriate stereotype for a user.

The classification base is also used to infer values for the inferable characteristics during the learning process. The variables of the conclusions of the classification neurules

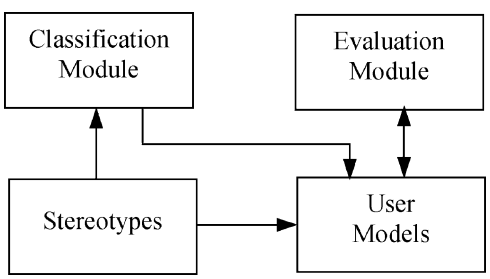

Fig. 3. The structure of the User Modeling Unit.

correspond to the inferable characteristics. The variables of the conditions correspond to the interaction parameters that the inferable characteristics are based on. For example, the concentration level is inferred based on the number of accesses to a concept units and the time spend on them. The user models are dynamically updated during the learning sessions.

Concept knowledge is a representation of which concepts and in what degree the user has already known (or learned). We use a combination of stereotypes and the overlay model (see, e.g. in Brusilovsky, 1998) to represent a user's *concept knowledge*. Stereotypes are used for initialisation purposes via the classification module. Stereotypes give values to subgroup knowledge level. The overlay model is related to the concept knowledge level.

The *evaluation module* evaluates the user's performance during a learning session, based on the interaction of the user with the system, and accordingly updates the user model. More specifically, based on the interaction parameters, it assigns values to the concepts the user has dealt with, denoting the knowledge-level of the user about those concepts. This is done by the *evaluation base*, a neurule base contained in the evaluation module.

One of the tasks of the evaluation neurules is to decide on the mark level a user achieved after having got through a test. The mark level ranges from 'low' to 'excellent'. The decision is based on the number of times the user asked for assistance, the number of related examples requested by the user and the number of answering attempts made by the user. The conditions of the neurules contain these parameters. Table 2 presents an example evaluation neurule (see Section 8 for the way it was produced). Mark levels concern concepts. Based on the mark levels of the concepts, the knowledge levels of the concept (sub)groups are derived.

5.2. Pedagogical unit

The pedagogical unit concerns the pedagogical decisions made during a learning session. It provides the knowledge infrastructure for tailoring the presentation of the teaching material according to the information contained in the user model. The pedagogical model consists of three knowledge bases: (a) method selection base; (b) concept selection base and (c) unit selection base. Each of these knowledge bases contains a number of neurules.

Table 2
An example evaluation neurule

(−9.7) if assistance-times is 1 (4.7),
assistance-times is 0 (4.6),
solution-attempts is 2 (4.6),
requested-examples is >1 (3.2),
requested-examples is 1 (1.4)
then mark is average

Table 3
An example neurule for the teaching method selection

(−20.2) if concentration-level is low (7.2),
knowledge-level is low (6.4),
accessed-units <0.25 (5.4),
def-teaching-method is theory-examples-exercises (2.5),
def-teaching-method is examples-exercises (2.4)
then teaching-method is theory-examples-exercises

The *method selection base* deals with the selection of the appropriate teaching method. Selection is based on parameters concerning the user model and the specific concept subgroup. Such parameters are the user's concentration level, knowledge level as well as the percentage of accessed course units (belonging to the specific concept subgroup). In addition, the concept subgroup's default teaching method is taken into account. These parameters appear in the conditions of the neurules used to select the teaching method. There are totally six teaching methods (see Section 7). For instance, according to one of them, in order to teach the user a specific concept sub-group, course units containing theory, examples and exercises should be involved. Another method states that, the most appropriate way of teaching would be to present only examples and exercises. Table 3 presents an example neurule for the teaching method selection.

The *concept selection base* is used to select the appropriate concepts, in order a user-adapted lesson plan be constructed. The selection is based on the user's prior concept knowledge, sub-domain knowledge level, desired detail level and the concepts' detail level. More specifically, for a specific sub-domain, the concepts for which the user's knowledge level is unsatisfactory are identified. These concepts are candidates for participating in the lesson plan. Concepts whose detail level is incompatible with the user's desired detail level are eliminated from the candidate set.

The *unit selection base* is used to select the course units, which are suitable for presentation. For this purpose, the student characteristics of the user model, the selected teaching method as well as the meta-description of the course units are taken into account.

5.3. Inference system

The inference system consists of two components: (a) fact base and (b) inference engine. The *fact base* contains facts related to the current user and learning session. Actually, all necessary information from the user model and domain knowledge is stored as facts in the fact base.

The *interface engine* (IE) implements the way neurules co-operate to reach conclusions. The inference process gives pre-eminence to symbolic reasoning and is based on a backward chaining strategy (see, e.g. in Gonzalez & Dankel, 1993). As soon as the initial input data is given, the neurules related to the final decisions (output neurules) are considered for evaluation. One of them is selected for

evaluation. Selection is based on textual order. A rule succeeds if the output of the corresponding adaline unit is computed to be '1', i.e its activation value a gets greater than zero ($a > 0$), after evaluation of its conditions (inputs).

A condition evaluates to 'true', if it matches a fact in the fact base, which means that there is a fact with the same variable, predicate and value. A condition evaluates to 'unknown', if there is a fact with the same variable, predicate and 'unknown' as its value. A condition cannot be evaluated if there is no fact in the fact base with the same variable. If the variable is inferable, however, a neurule with a conclusion containing that variable is examined. A condition evaluates to 'false', if there is a fact with the same variable, predicate and different value. Inference stops either when one or more output neurules are fired (success) or there is no further action (failure).

To illustrate how a neurule evaluation is made, we present the following example. Suppose that we have a knowledge base containing the four neurules of Table 12 (see Section 7.1.3) and a fact base with the following facts in it: {'assistance-times is 1', 'solution-attempts is >1', 'requested-examples is >1'}. IE starts examining each one of the rules, to find whether any of them fires (succeeds). To this end, the activation value a (see Section 2) of each rule is calculated (recall that the value of a condition is '1' if it is true and '0' if it is false), as follows (where terms corresponding to false conditions are omitted, since they give '0' as a result)

$$a_{PN1} = (-0.5) + 1 \times (-3.8) + 1 \times 2.8 + 1 \times (-0.70) = -5.0 + 2.8 = 0 - 2.2 < 0 \text{ (failure)}$$
$$a_{PN2} = (-1.0) + 1 \times (-4.1) + 1 \times (-2.2) + 1 \times (-0.4) = -7.7 < 0 \text{(failure)}$$
$$a_{PN3} = (-2.6) + 1 \times 4.7 + 1 \times (-1.0) + 1 \times (-0.2) = -3.8 + 4.7 = 0.9 > 0 \text{ (success)}$$

Since PN3 succeeds, the inference process stops and the conclusion 'mark-level is average' is produced and put in the fact base.

It is important to notice that even if there is no information in the fact base about a variable's value, evaluation can proceed and produce a result. For example, if 'required-examples is >1' is missing from the fact base above, evaluation proceeds as follows

$$a_{PN1} = (-0.5) + 1 \times (-3.8) + 1 \times 2.8 + 0.5 \times (-0.70) = -4.65 + 2.8 = -2.55 < 0 \text{ (failure)}$$
$$a_{PN2} = (-1.0) + 1 \times (-4.1) + 0.5 \times (-2.2) + 1 \times (-0.4) = -6.6 < 0 \text{ (failure)}$$
$$a_{PN3} = (-2.6) + 1 \times 4.7 + 1 \times (-1.0) + 0.5 \times (-0.2) = -3.7 + 4.7 = 1.0 > 0 \text{ (success)}$$

So, again NR3 succeeds and its conclusion is added to the fact base. This does not happen with symbolic rules, where evaluation stops in such cases.

Actually, the inference process is more complicated and some heuristics are used to make it more efficient not only than the symbolic rule-based one, but also than other similar efforts (Hatzilygeroudis & Prentzas, 2001b). For example, notice that for efficiency reasons the conditions of a neurule are ordered in an ascending way based on the absolute values of their significance factors.

6. System supervisor

The system supervisor controls the whole learning process. Its main objective is to construct a user-adapted lesson plan and dynamically update it during a learning session. A learning session concerns a specific concept group. The learning process consists of the following steps

(i) Select a concept subgroup
(ii) Select and order corresponding concepts
(iii) Select a teaching method
(iv) Select and order corresponding course units
(v) Evaluate user's performance and update user model
(vi) If knowledge level is not adequate, go to step (iii) or (iv). Otherwise, go to (i).

Fig. 4 depicts the overall learning process. In each step, the system supervisor calls the IE to act on the appropriate neurule base from the pedagogical and the user-modelling units. The results are put in the fact base and then propagated to the appropriate knowledge base. The system supervisor also takes care of ordering the selected concepts and the course units. Concepts selected via the concept selection base are ordered, based on the links between them. For example, concept 'HTML' will be put before 'Hypertext', since it is one of its prerequisites.

Ordering of the course units is primarily based on their pedagogical type and secondarily on their difficulty level. Ordering based on the pedagogical type is specified by

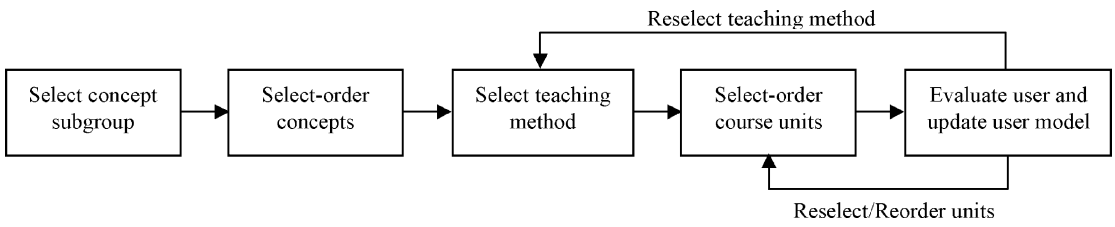

Fig. 4. Steps of the learning process.

the selected teaching method. Subsequent ordering is based on the difficulty level.

After the evaluation step, if the user's knowledge level about each concept belonging to the initial lesson plan is greater than or equal to concept's lowest acceptable knowledge level, another concept subgroup is selected and a new learning session ensues. Otherwise, the process goes back to step (iii) or (iv), causing reselection of the teaching method and/or course units and re-evaluation. The system records whether a teaching method has been successful or unsuccessful for a specific concept.

7. Knowledge management unit

The task of the *knowledge management unit* (KMU) is two-fold: (a) to acquire knowledge from various sources (e.g. experts, empirical data) and (b) to update knowledge stored in the knowledge bases of the ITS. Therefore, we distinguish two functional modes of the unit: *knowledge acquisition* mode and *knowledge update* mode. The overall structure of KMU is illustrated in Fig. 5.

7.1. Knowledge acquisition

In the knowledge acquisition mode, KMU interacts with an external source of knowledge (e.g. an expert) in order to elicit expert knowledge and represent it in the knowledge bases of the ITS. The objective of the knowledge acquisition mode is to finally produce a set of neurules that represent the expert knowledge. That knowledge can be acquired from three types of sources (see Fig. 5): (a) symbolic rules; (b) an expert and (c) empirical data.

7.1.1. From symbolic rules

In case (a), symbolic rules are converted into neurules via the *rules converting mechanism* (RCM). The symbolic rules may exist from another similar application or may have been elicited from an expert, in the traditional way (i.e. via interviews). The rules, before given as an input to RCM, should be transformed in the following format

if $C_1, C_2, \ldots, C_n$ **then** D

where the conditions C_i and the conclusion D have the same format as those of a neurule and ',' denotes the logical connective 'AND'.

The main idea in the RCM algorithm is to merge all symbolic rules with the same conclusion, called a *merger set*, into one neurule, i.e. an adaline neural unit (called the *merger* of the rules in the merger set), which, depending on the input values, will give the appropriate output. To this end, an initial neurule, having as conditions the conditions of all the symbolic rules in the merger set and random initial values for its factors, is produced. Then the truth table of the combined logical function of the rules is produced and refined (invalid combinations are eliminated). The refined combined truth table provides the necessary training knowledge patterns to train the neurule (i.e. calculate the bias and significance factors). Training is made via the well-known LMS algorithm. However, due to possible non-linearity of the patterns, this is not always feasible (see, e.g. Gallant, 1993). To overcome the problem, we split the initial merger set into two subsets and try to produce one neurule for each subset. If it is not possible for one or both subsets, we further split the non-linear subset(s) into two other subsets and so on until all subsets produce one neurule. So, finally more than one neurule are produced from the initial merger set. The way

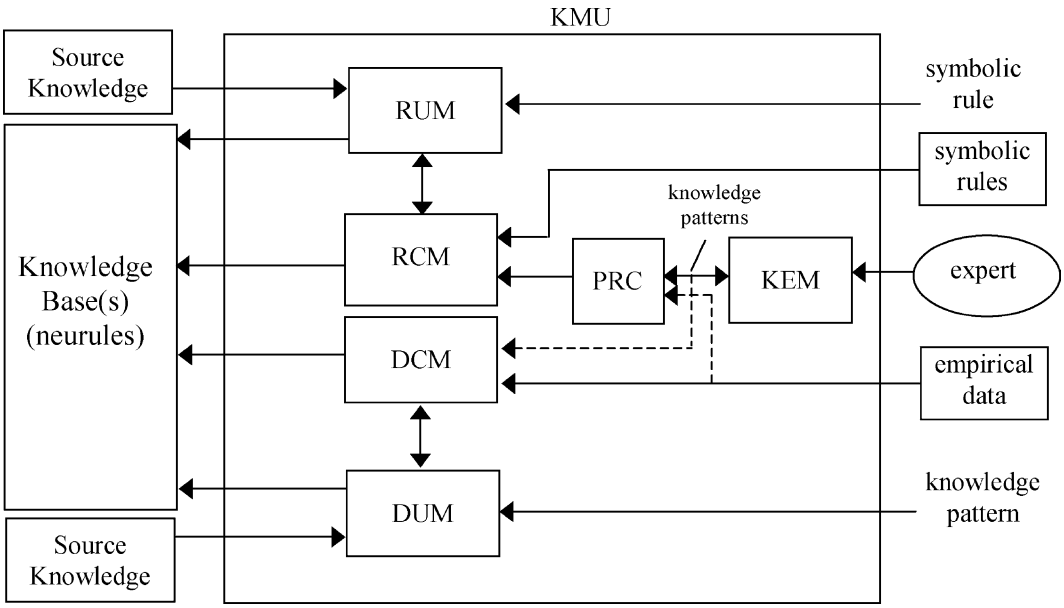

Fig. 5. The structure of the Knowledge Management Unit (KMU).

a subset is split is based on some strategy. The RCM algorithm is presented in detail in Hatzilygeroudis and Prentzas (2000). In any case, the number of the produced neurules is significantly less than that of the initial symbolic rules.

7.1.2. From an expert

In case (b), knowledge is elicited from an expert, via the *knowledge elicitation module* (KEM), in the form of *knowledge patterns*, which are then converted into symbolic rules, via the *pattern-to-rule conversion* (PRC) process, and finally into neurules, via the RCM algorithm.

A *knowledge pattern* relates a number of *variables* with a tuple of (their) values. In our case, variables represent user model characteristics. Each variable can take values from a set of *discrete values*. Each pattern has the following format

$$[v_1, v_2, \ldots, v_n]$$

where v_i $(i = 1, n)$ are values (numeric or symbolic) of the corresponding variables.

To produce knowledge patterns, *dependency information* is needed. Dependency information indicates which of the variables (intermediate or output variables) depend on which other variables (input or intermediate variables).

KEM interacts with the expert in order to assist him/her in creating knowledge patterns. As experienced, eliciting knowledge patterns from an expert is easier than directly eliciting symbolic rules.

The elicitation process is outlined below

1. Ask the expert to determine the input, intermediate and output variables as well as a set of discrete values for each of them.
2. Ask the expert to supply dependency information regarding the variables.
3. For each intermediate and output variable x,
   - 3.1. Create and present to the expert all valid combinations $[v_1\ v_2\ \ldots\ v_n]$ of the values of the variables x_i $(i = 1, n)$ that x depends on
   - 3.2. For each one of the combinations ask the expert to give the right value d of x, or mark the combination as invalid, so that a set p of valid knowledge patterns of the form $[v_1\ v_2\ \ldots\ v_n, d]$ is created for x.
   - 3.3. For each knowledge pattern in p produce the corresponding symbolic rule, via the PRC process, so that a set r of symbolic rules is produced.
   - 3.4. Convert the symbolic rules in r into neurules via the RCM algorithm.

The production of a symbolic rule from a knowledge pattern via the PRC process is quite straightforward. If $x_1, x_2, \ldots, x_n$ are the variables that x depends on and $[v_1, v_2, \ldots, v_n, d]$ is a knowledge pattern, the produced symbolic rule is

if x_1 is v_1,

x_2 is v_2,

...

x_n is v_n

then x is d

To illustrate the knowledge acquisition process, we present as an example the design of a part of the knowledge bases, related to the method selection base and the evaluation neurules[1].

1. In the first step, the expert (tutor) is asked to determine the input, intermediate and output variables of the application with their possible values. In our case the following are provided

Input variables:

(v1) solution-attempts (1, 2, > 2)
(v2) requested-examples (0, 1, >1)
(v3) assistance-times (0, 1, >1)
(v4) units-percentage (<25%, 25–50%, 50–75%, >75%)
(v5) access-times (0, 1, <4, > 3)
(v6) time-spent (<1, 1–3, 3–5, >5)
(v7) def-teaching-method (theory–example–exercise, example–theory–exercise, example–exercise, theory, example, exercise)

Intermediate variables:

(v8) knowledge-level (unknown, low, average, high)
(v9) concentration-level (unknown, low, average, high)
(v10) mark-level (low, average, high, excellent)

Output variables:

(v11) teaching-method (theory–example–exercise, example–theory–exercise, example–exercise, theory, example, exercise)

2. In the second step, the expert is asked to provide dependency information between input and intermediate as well as between intermediate and output variables. To this end, a table is presented and the expert is called to fill in the dependency information (by inserting an 'x' in the appropriate cell). This information is provided is Table 4.

3. In the third step, for each intermediate and output variable, a table, containing all combinations of the values of the variables it depends on, is presented. The expert is called then to discard invalid combinations and fill in

[1] For the sake of simplicity, the examples used in the paper are less complicated than the real cases.

Table 4
Dependency information given by the expert

	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10
Knowledge-level (v8)				x						x
Concentration-level (v9)					x	x				
Mark-level (v10)	x	x	x							
Teaching-method (v11)							x	x	x	

the right value of the (intermediate or output) variable for each valid combination. This step the intermediate variable 'mark-level' (with the last column completed by the expert) is presented in Table 5 (there are no invalid combinations).

After that, corresponding symbolic rules are created via the PRC process. For example, the symbolic rules created for the rows that have 'average' as value for 'mark-level' are presented in Table 6. There are eight (8) rules (one for each row).

These are then transformed into neurules via RCM. The produced neurules are presented in Table 7. Notice that only four (4) neurules are produced. Although they may have more conditions than the symbolic ones, they need notably less storage space. The produced neurules are equivalent to the symbolic ones, i.e. the same conclusions are produced for the same inputs.

There is an *alternative elicitation process* to the above. According to that, the knowledge patterns produced after step 3.2 are not converted into symbolic rules (step 3.3) and then into neurules (step 3.4), but are converted directly into neurules, via the DCM algorithm (see Section 7.1.3). This process is preferable in cases that the produced knowledge patterns are incomplete, that is they do not cover, for some reason, all or most of the possible valid combinations of the variables values (see Section 7.1.3). Thus, if we have a complete set of knowledge patterns, which usually is the case when eliciting an expert, we prefer to convert the produced knowledge patterns first into symbolic rules and then into neurules, which is called *indirect conversion*, and not directly, which is called *direct conversion* (this is why the dashed arrow from KEM output to DCM in Fig. 5).

Notice that the conditions of NR1 and NR4 are the same as those of R1 and R8. This means that {R1} and {R8} were

Table 5
Knowledge patterns created by the system and completed by the expert

Knowledge pattern no.	Solution-attempts (v1)	Requested-examples (v2)	Assistance-times (v3)	Mark-level (d)
1	1	0	0	Excellent
2	1	0	1	High
3	1	0	>1	High
4	1	1	0	Excellent
5	1	1	1	High
6	1	1	>1	High
7	1	>1	0	High
8	1	>1	1	High
9	1	>1	>1	Average
10	2	0	0	High
11	2	0	1	Average
12	2	0	>1	Average
13	2	1	0	Average
14	2	1	1	Average
15	2	1	>1	Low
16	2	>1	0	Average
17	2	>1	1	Average
18	2	>1	>1	Low
19	>2	0	0	Average
20	>2	0	1	Low
21	>2	0	>1	Low
22	>2	1	0	Low
23	>2	1	1	Low
24	>2	1	>1	Low
25	>2	>1	0	Low
26	>2	>1	1	Low
27	>2	>1	>1	Low

Table 6
Symbolic rules for the 'average' knowledge patterns

R1 if solution-attempts is 1, requested-examples is >1, assistance-times is >1 then mark is average	R5 if solution-attempts is 2, requested-examples is 1, assistance-times is 1 then mark is average
R2 if solution-attempts is 2, requested-examples is 0, assistance-times is 1 then mark is average	R6 if solution-attempts is 2, requested-examples is >1, assistance-times is 0 then mark is average
R3 if solution-attempts is 2, requested-examples is 0, assistance-times is >1 then mark is average	R7 if solution-attempts is 2, requested-examples is >1, assistance-times is 1 then mark is average
R4 if solution-attempts is 2, requested-examples is 1, assistance-times is 0 then mark is average	R8 if solution-attempts is >2, requested-examples is 0, assistance-times is 0 then mark is average

the merger subsets of NR1 and NR4, respectively. That is, they are actually transformations of individual symbolic rules into neurules. In contrast, NR2 is a merger of {R4, R6} as well as NR3 is a merger of {R2, R3, R5, R7}.

7.1.3. From empirical data

Finally, in case (c), empirical data is provided to the system (e.g. stored in a file) and converted into symbolic rules via the *data converting mechanism* (DCM). This may require dependency information between the variables that represent the empirical data, which is given by an expert.

Empirical data is data recorded from real cases that have been verified by objective means. Often, empirical data is extracted from observations (as in most pedagogical cases) and verified by a team of experts. Empirical data consists of a set of *knowledge patterns* in the form $[v_1, v_2, \ldots, v_n]$.

Based on dependency information, as many sets of knowledge patterns as the intermediate and the output variables are extracted from the initial set of patterns. Each pattern now has the form

$$[v_1, v_2, \ldots, v_m, d]$$

where d is the value of the corresponding intermediate or output variable and v_i $(i = 1 \ldots m, m \leq n)$ are values of the variables it depends on.

The acquisition algorithm can be outlined as follows

1. Ask the user to provide the source (e.g. a file) of the empirical data (i.e. variable names and knowledge patterns).
2. Ask the user to provide corresponding dependency information.
3. Based on the dependency information, produce the necessary data subsets.
4. For each data subset, produce the corresponding neurules via the DCM.

The DCM algorithm is presented in detail in (Hatzilygeroudis and Prentzas, 2001a). The main objective of the algorithm is to create one neurule for each data subset. However, due to possible non-linearity of the knowledge patterns of the subset, this is not always feasible. If this is the case, we split the initial subset into two other subsets and try to produce one neurule for each of the new subsets. If it is not possible, we split the non-linear set(s) into two other subsets and so on until all subsets produce one neurule. So, finally more than one neurule are produced. Some strategy is used in splitting a set into subsets.

For example, consider the empirical data set[2] (Table 8) introduced in the system by the user (step 1) and corresponding dependency information (step 2) shown in Table 9.

Dependency information shows that 'mark-level' and 'knowledge-level' are inferable variables, that is, their values depend on the values of other variables. More specifically, 'mark-level' can be considered as an intermediate variable, whereas 'knowledge-level' as an output variable.

Then, the empirical data *subsets* presented in Tables 10 and 11 are produced (step 3). Table 10 is related to the intermediate variable 'mark-level' (notice that the knowledge patterns are the same as patterns No. 2, 7, 9, 11, 13, 14, 18, 19, 20 and 24 of Table 5), whereas Table 11 to the output variable 'knowledge-level'.

In Table 12, the neurules produced from the 'mark-level' data subset (via the main DCM algorithm) are presented (step 4). Notice that two neurules are produced for the knowledge patterns related to 'average' value, whereas only one for each of 'high' and 'low' values. This means that the patterns related to 'average' value constitute a non-linear set, thus could not be covered by only one neurule (adaline unit).

There is an *alternative acquisition process* to the above. According to that, the knowledge patterns in the data subsets produced after step 3 are not converted directly into neurules (step 4), but they are first converted to symbolic rules via the PRC component and then to neurules via RCM, that is indirect conversion is used instead of direct. This process is preferable in cases that the produced knowledge patterns are complete, that is they cover all or most of the possible valid combinations of the values of the variables of the knowledge patterns.

This is due to the following fact. Experiments have shown that conversion of symbolic rules, representing

[2] Since there are no available real empirical data for this case and for reasons that will become clear later, we use an artificially produced data set, based on the patterns in Table 5.

Table 7
Neurules produced from the symbolic rules of Table 6

NR1
(−5.6) if solution-attempts is 1 (2.8),
requested-examples is >1 (1.5),
assistance-times is >1 (1.4)
then mark is average

NR2
(−12.9) if solution-attempts is 2 (6.5),
requested-examples is 0 (5.0),
assistance-times is 1 (4.6)
assistance-times is >1 (2.8)
then mark is average

NR3
(−13.6) if solution-attempts is 2 (8.2),
assistance-times is 1 (5.0),
requested-examples is 0 (4.4),
requested-examples is 1 (1.6),
assistance-times is >1 (1.2),
requested-examples is >1 (0.9)
then mark is average

NR4
(−6.1) if solution-attempts is >2 (5.0),
assistance-times is 0 (1.1),
requested-examples is 0 (1.0)
then mark is average

Table 8
Empirical data set

Solution-attempts	Requested-examples	Assistance-times	Units-percentage	Mark-level	Knowledge-level
1	0	1	>0.75	High	High
2	1	1	<0.25	Average	Low
1	>1	0	<0.50	High	Average
2	0	1	>0.75	Average	High
>2	1	>1	<0.25	Low	Low
1	>1	>1	<0.50	Average	Average
>2	0	1	<0.50	Low	Low
2	1	0	<0.50	Average	Average
2	>1	>1	<0.25	Low	Low
>2	0	0	<0.50	Average	Average

Table 9
Dependency information for the empirical data set of Table 8

	Solution-attempts	Requested-examples	Assistance-times	Units-percentage	Mark-level
Mark-level	x	x	x		
Knowledge-level				x	x

a number of knowledge patterns, into neurules, in the above case, produces a smaller knowledge base (less neurules with less conditions) than the conversion of the same knowledge patterns directly into neurules. However, direct conversion produces neurules with generalisation capabilities, whereas indirect one produces neurules that actually have no generalisation capabilities. This means that neurules from direct conversion may produce the right result even if input values belong to unknown knowledge patterns, that is patterns not used for their production (training). Thus, if we have an incomplete set of knowledge patterns, which is usually the case with empirical data, we prefer to convert the given knowledge patterns directly into neurules and not indirectly (this is why the dashed arrow to PRC in Fig. 5).

To illustrate this, let's convert the knowledge patterns of the data subset of Table 10 into symbolic rules (via PRC) and then into neurules (via RCM). The produced neurules are presented in Table 13.

Notice that (a) the neurules of Table 13 are more than those in Table 12[3] and (b) if values of 'unknown' knowledge patterns come as inputs to the neurules, they do not produce any right result. For example, if we consider the complete set of patterns (Table 5), the neurules of Table 13 do not correctly classify (i.e. produce a correct result for) any of the patterns that do not belong to the subset corresponding to Table 10. However, the neurules of Table 12 can correctly classify patterns No. 3, 5, 8, 15, 21, 23, 26, 27 (that is eight out of 16 'uknown' patterns).

7.2. Knowledge update

In the knowledge update mode, KMU deals with knowledge refinements or changes to the knowledge bases

[3] This happens because the data subset is incomplete, otherwise they would be less. Notice, however, that even now the overall conditions of the symbolic rules in Table 13 are less than those of the neurules in Table 12.

Table 10
Mark-level data subset

Solution-attempts	Requested-examples	Assistance-times	Mark-level
1	0	1	High
2	1	1	Average
1	>1	0	High
2	0	1	Average
>2	1	>1	Low
1	>1	>1	Average
>2	0	1	Low
2	1	0	Average
2	>1	>1	Low
>2	0	0	Average

Table 11
Knowledge-level data subset

Units-percentage	Mark-level	Knowledge-level
>0.75	High	High
<0.25	Average	Low
<0.50	High	Average
>0.75	Average	High
<0.25	Low	Low
<0.50	Average	Average
<0.50	Low	Low
<0.50	Average	Average
<0.25	Low	Low
<0.50	Average	Average

of the ITS. Knowledge update refers to knowledge refinements or changes either during the last steps in the construction stage or during the operation and maintenance stage. Most knowledge updates usually take place during the construction stage, when the system prototype is implemented and tested. This happens because, e.g. the knowledge about teaching methods or student evaluation methods is rather difficult to be acquired and represented in a computer-based system and many refinements are required. Also, the operation of the system and the consequent feedback from users can also spark off changes to the knowledge.

In contrast to other hybrid approaches (like, e.g. the one used in Magoulas, Papanikolaou, & Grigoriadou, 2001), neurules offer methods for easy knowledge update. The objective of the methods is to make the required changes with as little reconstruction (retraining) of a knowledge base as possible. This is achieved thanks to the modular nature of neurules and the ideas used in the methods. Recall that a neurule base is constructed from either a set of symbolic rules or a set of knowledge patterns. We call either set the *source knowledge* of a neurule base. For example, the symbolic rules of Table 6 constitute the source knowledge of the neurule base of Table 7. Also, the knowledge patterns of Table 10 is the source knowledge of the neurule base of Table 12.

There are methods for both types of source knowledge. However, to apply the methods, source knowledge should be stored in the system (see Fig. 5). Based on the type of source knowledge, we distinguish two modes of knowledge update: (a) the *rule-based update mode*, when source knowledge is actually a set of symbolic rules, and (b) the *pattern-based update mode*, when source knowledge is a set of knowledge patterns.

7.2.1. Rule-based update

The rule-based update mode concerns knowledge acquired from (a) existing symbolic rules; (b) an expert

Table 12
Neurules directly produced from the knowledge patterns of Table 10

PN1
(−0.5) if assistance-times is 0 (−4.6),
solution-attempts is 1 (−3.8),
solution-attempts is >2 (3.0),
assistance-times is >1 (2.8),
requested-examples is 0 (−2.0),
requested-examples is 1 (0.90),
requested-examples is >1 (−0.70),
solution-attempts is 2 (−0.30),
assistance-times is 1 (−0.30)
then mark-level is low

PN2
(−1.0) if assistance-times is >1 (−4.1),
solution-attempts is >2 (−2.8),
requested-examples is >1 (−2.2),
assistance-times is 1 (1.6),
requested-examples is 1 (1.5),
solution-attempts is 2 (1.3),
assistance-times is 0 (1.3),
requested-examples is 0 (−0.6),
solution-attempts is 1 (−0.4)
then mark-level is average

PN3
(−2.6) if solution-attempts is 2 (−6.2),
requested-examples is 1 (−6.0),
assistance-times is 1 (−5.7),
assistance-times is >1 (4.7),
requested-examples is 0 (3.2),
assistance-times is 0 (−2.7),
solution-attempts is >2 (2.6),
solution-attempts is 1 (−1.0),
requested-examples is >1 (−0.2)
then mark-level is average

PN4
(−0.8) if solution-attempts is 1 (3.2),
assistance-times is >1 (−2.7),
solution-attempts is 2 (−2.6),
requested-examples is 1 (−2.6),
solution-attempts is >2 (−2.5),
requested-examples is 0 (1.4),
assistance-times is 1 (1.0),
requested-examples is >1 (−0.7),
assistance-times is 0 (−0.3)
then mark-level is high

Table 13
Neurules indirectly produced from the knowledge patterns of Table 10

SN1
(−6.1) if solution-attempts is 1 (4.4),
requested-examples is 0 (1.5),
assistance-times is 1 (1.0)
then mark-level is high

SN2
(−5.6) if solution-attempts is 1 (2.8),
requested-examples is >1 (1.5),
assistance-times is 0 (1.4)
then mark-level is high

SN3
(−13.2) if solution-attempts is >2 (6.9),
requested-examples is 0 (5.0),
assistance-times is 0 (3.1)
then mark-level is average

SN4
(−11.7) if solution-attempts is 2 (6.9),
assistance-times is 1 (4.a6),
requested-examples is 1 (2.9),
assistance-times is 0 (2.8),
requested-examples is 0 (1.4)
then mark-level is average

SN5
(−6.1) if solution-attempts is 1 (5.0),
assistance-times is >1 (1.1),
requested-examples is >1 (1.0)
then mark-level is average

SN6
(−9.8) if solution-attempts is >2 (4.9),
requested-examples is 1 (3.2),
assistance-times is >1 (2.6)
then mark-level is low

SN7
(−6.3) if solution-attempts is >2 (4.8),
requested-examples is 0 (1.3),
assistance-times is 1 (1.2)
then mark-level is low

SN8
(−9.6) if solution-attempts is 2 (4.9),
requested-examples is >1 (4.4),
assistance-times is >1 (1.3)
then mark-level is low

(indirect conversion case) and (c) empirical data (indirect conversion case).

The updates, in this mode, should satisfy the following requirements: (a) enhance source knowledge with a new symbolic rule, (b) modify an existing symbolic rule in source knowledge and (c) remove an existing symbolic rule from source knowledge (because, e.g. it is experienced to be invalid). There are two algorithms to satisfy the above requirements: (a) the *rule insertion algorithm*, for inserting a new rule, and (b) the *rule removal algorithm*, for removing an existing rule. Modification of a rule results from the combination of removing the existing rule and inserting the new one (i.e. the modified).

The main idea of the rule insertion algorithm is to find the smallest merger (sub)set the rule to be inserted is closer to. The 'closeness' of the rule basically depends on the number of common conditions of the rule and the rules in the merger (sub)set. When it is found, the neurules produced from the (sub)set are removed from the current knowledge base, the rule is inserted into the merger (sub)set and the new (sub)set is transformed into neurules, via the RCM, which are then inserted into the knowledge base.

The main idea of the rule removal algorithm is to find the merger (sub)set the rule to be removed belongs to. When it is found, the neurules produced from the (sub)set are removed from the current knowledge base, the rule is removed from the merger (sub)set and the new (sub)set is transformed into neurules, via the RCM, which are then inserted into the knowledge base.

To achieve their objectives, both algorithms use a tree structure, called the 'splitting tree', which stores information related to merger set splitting (this information is stored in the corresponding source knowledge components). Both algorithms are presented in detail in (Prentzas & Hatzilygeroudis, 2002).

The *rule update module* (RUM) implements those algorithms and interacts with RCM and the corresponding knowledge base and source knowledge to achieve knowledge update. It accepts as input a symbolic rule and results in changes to the corresponding knowledge base. The rule-based update process is as follows

1. Ask the user to specify the type of update action (insertion, removal)
2. Ask the user to provide the knowledge item related to the update
3. If it is a knowledge pattern, convert it into a symbolic rule via RCM
4. Apply the rule insertion or rule removal algorithm, according to the user choice in step 1.

For example, if symbolic rule R8 is to be removed from the source knowledge of Table 6, the only neurule that is affected is NR4, because its merger subset is {R8}. So, NR4 is just removed. If rule R7 is to be removed, NR3 is affected, since its merger subset is {R2, R3, R5, R7}. Then, NR3 is removed from the neurule base, R7 is removed from the merger subset and the remaining set ({R2, R3, R5}) is used to produce its corresponding neurule, which is presented in Table 14.

Suppose now, for the sake of simplicity, that the old rule R7 is to be added to the neurule base. Again, only neurule

Table 14
Revised neurule NR3

NR3
(−17.2) if solution-attempts is 2 (10.0), assistance-times is 1 (6.8), requested-examples is 0 (6.2), assistance-times is >1 (3.0), requested-examples is 1 (1.6) then mark is average

NR3 is affected, because R7 is closer to its merger subset. R7 is inserted into the source knowledge and the merger subset, NR3 is removed from the neurule base and a new neurule is produced from the new merger subset (actually NR3 in Table 6) and added to the neurule base.

7.2.2. Pattern-based update

The pattern-based update mode deals with knowledge acquired from (a) empirical data (direct conversion case) and (b) an expert (direct conversion case).

In this mode, a basic requirement to be satisfied is the enhancement of source knowledge with a new knowledge pattern. Removal of a knowledge pattern is not an actual need for case (a), since empirical data is always verified. It could be for case (b). However, case (b) is not a usual case. On the other hand, a method to avoid reconstruction of all the relevant neurules, in the case of a pattern removal, seems to be technically difficult and a matter of further research.

So, there is one algorithm to satisfy the above requirement, the *pattern insertion algorithm*, for inserting a new pattern.

The main idea of the pattern insertion algorithm is to find the data (sub)set the pattern to be inserted is closer to. The 'closeness' of the pattern basically depends on the number of common values of the pattern and the patterns in the subset. When it is found, the neurules produced from the subset are removed from the current knowledge base, the pattern is inserted into the subset and the new subset is transformed into neurules, via DCM, which are then inserted into the knowledge base.

Again, the structure of 'splitting tree' is used, to achieve as little reconstruction of the neurules as possible. The algorithm is presented in detail in Prentzas, Hatzilygeroudis, & Tsakalidis (2002)).

The *data update module* (DUM) implements the above algorithm and interacts with DCM and the corresponding knowledge base to achieve knowledge update. It accepts as input a knowledge pattern and results in changes to the corresponding knowledge base. The pattern-based update process is as follows

1. Ask the user to provide the knowledge pattern
2. Apply the pattern insertion algorithm.

For example, suppose that the new knowledge pattern [2, 0, >1, average] is to be added to the source knowledge of Table 10. Then, neurule PN3 in Table 12 is only affected, because the new pattern is closer to its pattern subset ({[1, >1, >1, average], [>2, 0, 0, average]}). So, the new pattern is added to the source knowledge and the pattern subset, PN3 is removed and a new neurule (presented in Table 15) is produced and added to the neurule base.

Table 15
Revised neurule PN3

PN3
(−0.8) if assistance-times is 1 (−5.7), requested-examples is 0 (5.0), solution-attempts is 2 (−4.4), requested-examples is 1 (−4.2), assistance-times is >1 (2.9), requested-examples is >1 (−2.0), assistance-times is 0 (0.9), solution-attempts is 1 (0.8), solution-attempts is >2 (0.8) then mark-level is average

8. Benefits of using neurules

The use of neurules to represent knowledge in the ITS has revealed a number of benefits that they can offer to the construction of an ITS as well as its operation and maintenance More specifically:

(a) Neurules support incremental development of neurule-bases, because they retain the naturalness and modularity of symbolic rules. One can easily add new neurules or remove old neurules from a neurule base without making any other changes to it, given that they do not affect existing knowledge, because neurules are functionally independent units. This is difficult for other hybrid approaches.

(b) Neurules can be acquired in a semi-automated way from various sources, such as symbolic rules, empirical data or an expert, thus enabling exploitation of alternative knowledge sources. So, non-availability of a specific knowledge source (e.g. experts) can be overcome. This is very important for ITSs, because knowledge acquisition for them is harder than for conventional expert systems, due to the existence of more than one knowledge-based modules.

(c) Neurules are space-efficient. As it is proved from the examples in this paper and elsewhere (Hatzilygeroudis and Prentzas, 2000), neurules produce quite smaller knowledge bases compared to classical symbolic rules. The size reduction in the ITS is 35–40%.

(d) Neurule bases can be easily updated, i.e. without thorough reconstruction of them. This is quite helpful during both the construction and maintenance stage. Due to the nature of an ITS, many knowledge base

updates are required during the construction stage, when the system prototype is tested. Also, after thoroughly checking the effectiveness of the system by its interactions with end-users, further changes and updates of the incorporated expert knowledge may be required. Knowledge base updates constitute a bottleneck for other hybrid approaches, such as neuro-fuzzy ones.

(e) Neurules can make robust inferences. In contrast to symbolic rules, neurules can derive conclusions from partially known inputs. This is due to the fact that neurules integrate a connectionist component (adaline). This feature is useful, because, during a teaching session, certain parameters related to the user may be unknown.

(f) Neurules provide a time-efficient inference engine. This means that they require fewer computations compared to classical symbolic rules and even to other hybrid approaches in order to derive the same conclusions (Hatzilygeroudis and Prentzas, 2000, 2001b). This is very important, since an ITS is a highly interactive knowledge-based system requiring time-efficient responses to users' actions. Furthermore, the Web imposes additional time constraints.

9. Conclusions and future work

In this paper, we present the architecture and describe the functionality of a Web-based ITS, which uses a hybrid rule-based formalism, namely *neurules*, for knowledge representation. The system's pedagogical decisions are made by an expert system using neurules as knowledge representation formalism and its corresponding inference mechanism.

The use of neurules instead of symbolic rules or other hybrid neuro-symbolic approaches offers a number of advantages. Neurules are space and time efficient and offer a robust inference mechanism. Additionally, neurule bases can be incrementally constructed and easily updated. Finally, alternative available knowledge sources can be exploited for acquiring knowledge in a semi-automated way.

These last benefits give the ITS knowledge authoring capabilities. ITSs are systems that need these capabilities, because pedagogical knowledge is not static, but dynamic and changes are quite often required.

The use of hybrid approaches in ITSs is likely to gain interest in the following years. However, neurules have two weak points. First, they cannot represent fuzziness, which is necessary in some cases, e.g. for representing user model knowledge. To remedy this weakness, capabilities for fuzzy representation should be incorporated into neurules, which is one of our current efforts. Second, neurules cannot represent structural knowledge, like the one needed in domain knowledge. This kind of knowledge can be easily represented by other representation formalisms, such as semantic nets or conceptual graphs. Therefore, it seems that a multi-paradigm environment would be adequate for all knowledge representation requirements of an ITS.

References

Angelides, M., & Garcia, I. (1993). Towards an intelligent knowledge based tutoring system for foreign language learning. *Journal of Computing and Information Technology*, *1*, 15–28.

Brusilovsky, P. (1998). Methods and techniques of adaptive hypermedia. In P. Brusilovsky, A. Kobsa, & J. Vassileva (Eds.), *Adaptive hypertext and hypermedia*. Dordrecht: Kluwer Academic Publishers.

Brusilovsky, P., Schwarz, E., & Weber, G. (1996). ELM-ART: An intelligent tutoring system on World Wide Web. In C. Frasson, G. Gauthier, & A. Lesgold (Eds.), *Third International Conference on Intelligent Tutoring System-ITS 1996* (*Vol. 1086*) (pp. 261–269). *LNCS*, Berlin: Springer.

El-Khouly, M. M., Far, B. H., & Koono, Z. (2000). Expert tutoring system for teaching computer programming languages. *Expert Systems with Applications*, *18*, 27–32.

Gallant, S. I. (1993). Neural network learning and expert systems. Cambridge, MA: MIT Press.

Georgouli, K. (2002). The Design of a motivating intelligent assessment system. In S. A. Cerri, G. Gouarderes, & F. Paraguacu (Eds.), *Sixth International Conference, ITS-2002* (*Vol. 2363*) (pp. 261–269). *LNCS*, Berlin: Springer.

Gonzalez, A. J., & Dankel, D. D. (1993). The engineering of knowledge-based systems, theory and practice. Englewood Cliffs, NJ: Prentice Hall.

Hatzilygeroudis, I., & Prentzas, J. (2000). Neurules: improving the performance of symbolic rules. *International Journal on Artificial Intelligence Tools*, *9*, 113–130.

Hatzilygeroudis, I., & Prentzas, J. (2001a). Constructing modular hybrid knowledge bases for expert systems. *International Journal on Artificial Intelligence Tools*, *10*, 87–105.

Hatzilygeroudis, I., & Prentzas, J. (2001b). *An efficient hybrid rule based inference engine with explanation capability*. *Proceedings of the 14th International FLAIRS Conference*, Menlo Park, CA: AAAI Press, (pp. 227–231).

Hwang, G.-J. (1998). A tutoring strategy supporting system for distance learning on computer networks. *IEEE Transactions on Education*, *41*, 343–361.

Hwang, G.-J. (2003). A conceptual map model for developing intelligent tutoring systems. *Computers and Education*, *40*, 217–235.

Josephina, M. P., & Nkambou, R. (2002). Hierarchical representation and evaluation of the student in an intelligent tutoring system. In A. S. Cerri, G. Gouarderes, & F. Paraguacu (Eds.), *Proceedings of the Sixth International Conference on Intelligent Tutoring Systems—ITS 2002* (*Vol. 2363*) (pp. 708–717). *LNCS*, Berlin: Springer.

Magoulas, G. D., Papanikolaou, K. A., & Grigoriadou, M. (2001). Neuro-fuzzy synergism for planning the content in a web-based course. *Informatica*, *25*, 39–48.

Medsker, L. R. (1995). Hybrid intelligent systems. Dordrecht: Kluwer Academic Publishers.

Moundridou, M., & Virvou, M. (2003). Analysis and design of a Web-based authoring tool generating intelligent tutoring systems. *Computers and Education*, *40*, 157–181.

Nauck, D., Klawonn, F., & Kruse, R. (1997). Foundations of neuro-fuzzy systems. New York: Wiley.

Nkambou, R. (1997). *Using fuzzy logic in its-course generation*. *Proceedings of the Ninth IEEE International Conference on Tools with Artificial Intelligence*, Silver Spring, MD: IEEE Computer Society, (pp. 190–193).

Polson, M. C., & Richardson, J. J. (1988). Foundations of intelligent tutoring systems. London: Lawrence Erlbaum.

Prentzas, J., & Hatzilygeroudis, I. (2002). *Updating a hybrid rule base with changes to its symbolic source knowledge*. *Proceedings of the ECAI-2002, Lyon, France*, Amsterdam: IOS Press, (pp. 250–254).

Prentzas, J., Hatzilygeroudis, I., & Garofalakis, J. (2002). A web-based intelligent tutoring system using hybrid rules as its representational basis. In S. A. Cerri, & G. Gouarderes (Eds.), *Sixth International Conference, ITS-2002* (*Vol. 2363*). *LNCS*, Berlin: Springer.

Prentzas, J., Hatzilygeroudis, I., & Tsakalidis, A. (2002). *Updating a hybrid rule base with new empirical source knowledge*. *Proceedings of the 14th IEEE International Conference on AI Tools (ICTAI-02), Washington, DC, USA*, Silver Spring, MD: IEEE Computer Society Press.

Rich, E. (1989). Stereotypes and user modelling. In A. Kobsa, & W. Wahlster (Eds.), *User models in dialog systems* (pp. 35–51). Berlin: Springer.

Shiri, M. E., Aimeur, E., & Frassen, C. (1998). Modeling by case-based reasoning. In B. P. Goettl, H. M. Halff, C. L. Redfield, & V. J. Shute (Eds.), *Fourth International Conference on Intelligent Tutoring Systems-ITS 1998* (*Vol. 1452*) (pp. 394–404). *LNCS*, Berlin: Springer.

Simic, G., & Devedzic, V. (2003). Building an intelligent system using modern Internet technologies. *Expert Systems with Applications*, *25*, 231–246.

Stern, M., & Woolf, B. (1998). Curriculum sequencing in a web-based tutor. In B. P. Goettl, H. M. Halff, C. L. Redfield, & V. J. Shute (Eds.), *Fourth International Conference on Intelligent Tutoring Systems* (*Vol. 1452*). *LNCS*, Berlin: Springer.

Sun, R., & Alexandre, E. (Eds.), (1997). *Connectionist-symbolic integration: From unified to hybrid approaches*. London: Lawrence Erlbaum.

Urretavizcaya-Loinaz, M., & Fernandez de Castro, I. (2002). Artificial intelligence and education: an overview. *Upgrade, III*, (5), 53–58.

Vassileva, J. (1997). Dynamic courseware generation. *Journal of Computing and Information Technology*, *5*, 87–102.

Zhendong, V. K. (2001). Bayesian student modelling, user interfaces and feedback: a sensitivity analysis. *International Journal of Artificial Intelligence in Education*, *12*(2), 155–184.