Authoring Constraint-based Tutors in ASPIRE

Antonija Mitrovic, Pramuditha Suraweera, Brent Martin,
Konstantin Zakharov, Nancy Milik and Jay Holland

Intelligent Computer Tutoring Group
University of Canterbury, Christchurch, New Zealand
{tanja, psu16, brent, kza10, nmi14, jah130}@cosc.canterbury.ac.nz

Abstract: This paper presents a project the goal of which is to develop ASPIRE, a complete authoring and deployment environment for constraint-based intelligent tutoring systems (ITSs). ASPIRE is based on our previous work on constraint-based tutors and WETAS, the tutoring shell. ASPIRE consists of the authoring server (ASPIRE-Author), which enables domain experts to easily develop new constraint-base tutors, and a tutoring server (ASPIRE-Tutor), which deploys the developed systems. Preliminary evaluation shows that ASPIRE is successful in producing domain models, but more thorough evaluation is planned.

1 Introduction

Building a constraint-based tutor, like any other ITS, is a labour-intensive process that requires expertise in constraint-based modelling (CBM) and programming. While ITSs contain a few modules that are domain-independent, their domain model, which consumes the majority of the development effort, is unique. Our goal is to reduce the time and effort required for producing ITSs by building an authoring system that can generate the domain model with the assistance of a domain expert and produce a fully functional system. We also envisage that the authoring system would enable teachers, with little or no expertise in CBM, to build their own ITSs.

This paper presents ASPIRE, an authoring system that assists in the process of composing domain models for constraint-based tutors and automatically serves tutoring systems on the web. The proposed system is an enhancement of WETAS [4, 5], a web-based tutoring shell that facilitates building constraint-based tutors. WETAS is a prototype system that provides all the domain-independent components for text-based ITSs. The main limitation of WETAS is its lack of support for authoring domain models. ASPIRE guides the author through a semi-automated process for building the domain model and seamlessly deploys the resulting domain model to produce a fully functional web-based tutoring system.

The paper commences with a brief introduction to related authoring systems for building ITSs. Section 3 details the ASPIRE authoring system, including an outline of the domain authoring process and the architecture of the system. We also include an

overview the constraint generation algorithms, the central component of the authoring process. Finally, Section 4 presents conclusions and the directions of future work.

2 Related Work

Murray [10] classified ITS authoring tools into two main groups: pedagogy-oriented and performance-oriented. Pedagogy-oriented systems focus on instructional sequencing and teach relatively fixed content. On the other hand, performance-oriented systems focus on providing rich learning environments, where students learn by solving problems while receiving dynamic feedback on their progress. These systems have a deep model of expertise, which enables the tutor to correct the student as well as provide assistance on problem solving. Authoring systems thus need to support the acquisition of domain models. Typically, sophisticated machine learning techniques are used for acquiring domain rules with the assistance of a domain expert.

Only a few authoring systems are capable of generating domain models. Disciple, developed by Tecuci and co-workers [15, 16], is an example of a learning agent shell for developing intelligent educational agents. A domain expert teaches the agent to perform domain-specific tasks, similar to a manner of an expert teaching an apprentice, by providing examples and explanations. The expert is also required to supervise and correct the behaviour of the agent. Disciple acquires knowledge using a collection of complementary learning methods including inductive learning from examples, explanation-based learning, learning by analogy and learning by experimentation. A completed Disciple agent can be used to interact and guide students in performing tasks of the domain.

The Cognitive Tutor Authoring Tools (CTAT) [1, 2] assist in the creation and delivery of ITSs based on model tracing. The main goal of these tools is to reduce the amount of artificial intelligence (AI) programming expertise required. The system allows authors to create two types of tutors: ‘Cognitive tutors’ and ‘Pseudo tutors’. ‘Cognitive tutors’ contain a cognitive model that simulates the student's thinking to monitor and provide pedagogical assistance during problem solving. In contrast, ‘Pseudo tutors’ do not contain a cognitive model: to develop a tutor of this kind, the author needs to specify a recording of possible student actions and corresponding feedback messages. Although ‘Pseudo tutors’ do not require AI programming, they are specific to the demonstrated set of problems, and cannot deal with student actions’ which are not pre-specified by the author.

3 ASPIRE

ASPIRE assists with the creation and delivery of constraint-based tutoring systems. It generates constraints that make up the domain model with the assistance of the domain expert, minimising the programming expertise required for developing a new constraint-based tutor. The system also provides all the domain-independent functionality of constraint-based ITSs.

3.1 Authoring Process

Authoring a constraint-based tutor in ASPIRE is a semi-automated process, carried out with the assistance of the domain expert. The authoring process, summarised in Figure 1, consists of nine distinct phases. Initially, the author specifies general features of the chosen instructional domain, such as whether the domain consists of a sub-domains focusing on specific areas, and whether the domain is procedural or not. In the case of procedural domains, the author is required to enumerate the problem-solving steps. As an example, let us consider the procedural domain of adding fractions. The problem-solving procedure can be broken down into four steps, as outlined in Figure 2. Initially, it is necessary to check whether the two fractions have the same denominator; if that is not the case, the lowest common denominator must be found. Step two involves modifying the two fractions to have the lowest common denominator (when needed). After that, the two fractions are added, which may result in an improper fraction. Finally, the result is to be simplified, if appropriate.

1. Specifying the domain characteristics
2. Composing the domain ontology
3. Modelling the problem and solution structures
4. Designing the student interface
5. Adding problems and solutions
6. Generating syntax constraints
7. Generating semantic constraints
8. Validating the generated constraints
9. Deploying the tutoring system

Figure 1. The phases of the authoring process

In the second phase, the author develops an ontology of the chosen instructional domain, which plays a central role in the authoring process. ASPIRE-Author provides an ontology workspace for visually modelling ontologies (Figure 3). A domain ontology describes the domain by identifying important concepts and relationships between them. The ontology outlines the hierarchical structure of the domain in terms of sub- and super-concepts. Each concept might have a number of properties, and may be related to many other domain concepts. A preliminary study conducted to evaluate the role of ontologies in manually composing a constraint base showed that constructing a domain ontology assisted the composition of constraints [13]. The study showed that ontologies support authors to reflect on the domain, organise constraints into meaningful categories and produce more complete constraint bases.

1. Find the lowest common denominator (LCD)
2. Convert fractions to LCD as denominator
3. Add the resulting fractions
4. Simplify the final result

Figure 2. Problem-solving procedure for fraction addition

An ontology for the domain of adding fractions is illustrated in Figure 3. It contains *Number* as the most generic concept, which has two specialisations, *Whole-*

number and *Fraction*. *Whole-number* is further specialised into lowest common denominator (*LCD*), while *Fraction* is specialised into *Improper* and *Reduced*. The specialization/generalization relationships between domain concepts are visually represented as arrows between concepts. Figure 3 shows three additional relationships defined for the *Reduced Fraction* concept: *whole number*, *numerator* and *denominator*. While *numerator* and *denominator* and mandatory relationships, *whole number* may only occur if the resulting fraction needs to be simplified.

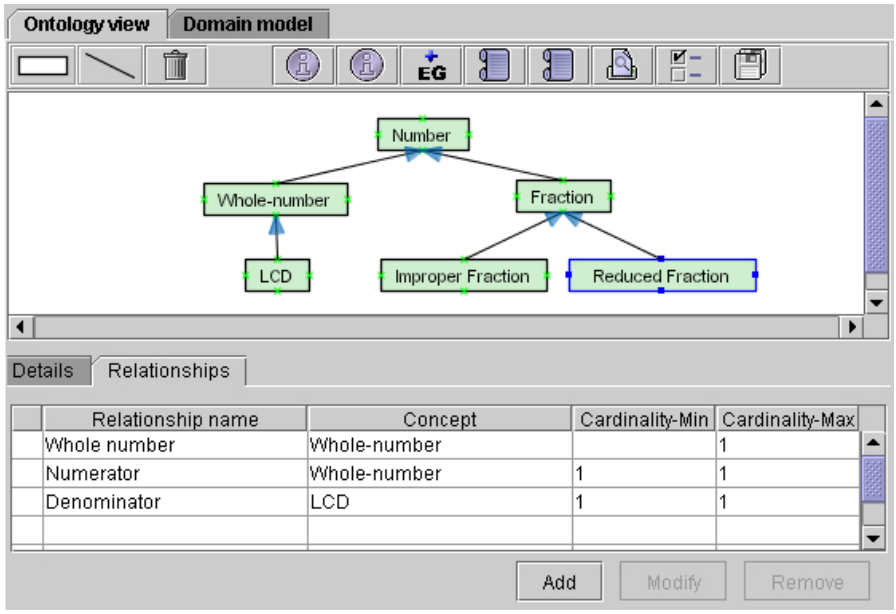

Figure 3. Ontology for adding fractions

In the third phase, the author specifies the problem/solution structures. Problems can consist of components (textual or graphical) and a problem statement. In our example domain, problems contain a common statement ("Add these two fractions"), and the problem to be solved (e.g. "1/3 + 1/5"). Student solutions may also consist of several components. The overall structure of solutions depends on whether the domain is procedural or declarative. A declarative task requires a single solution that may consist of a number of components, whereas a procedural task requires a solution for each step of the procedure. As the result, the structure of solutions for each step has to be modelled. The solution structure for fraction addition is outlined in Figure 4, showing also the corresponding domain concepts.

The student interface needs to be designed next. The final outcome of this phase is a form-based interface that can be used by students to compose their solutions. The system initially generates a default interface, placing an input area for each component defined in the solution structure [9]. The domain expert can rearrange the interface components in order to provide a more intuitive interface for students. An example of an interface for adding fractions is shown in Figure 5.

After designing the student interface, the author enters example problems and their solutions. For each problem, the author enters a problem statement, and one or

more correct solutions. In order for the authoring system to learn about different ways of solving a problem, the expert is required to provide multiple solutions to a problem depicting different ways of solving it. These solutions are used by the authoring system for generating semantic constraints.

Problem solving step	Solution component	Concept
1. Find LCD	LCD	LCD
2. Convert fractions to LCD	Fraction 1 numerator Fraction 1 denominator Fraction 2 numerator Fraction 2 denominator	Improper fraction
3. Sum of improper fractions	Improper sum numerator Improper sum denominator	Improper fraction
4. Final reduced sum	Final sum whole number Final sum numerator Final sum denominator	Reduced fraction

Figure 4. Solution structure for adding two fractions

Once example problems and their solutions are available, ASPIRE-Author generates the domain model. The syntax constraint generator analyses the domain ontology and generates syntax constraints directly from it. These constraints are generated by translating the restrictions on the properties and relationships of concepts specified in the ontology, as detailed in Section 3.3. The constraint generator produces an extra set of syntax constraints for procedural domains that ensure that the student progresses correctly in the problem solving process.

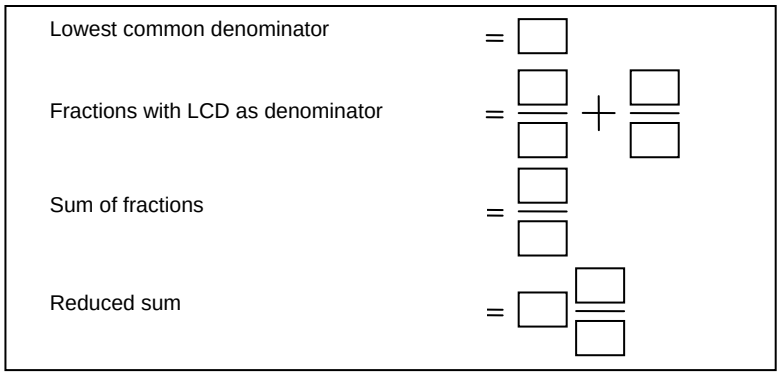

Figure 5. Student interface for adding two fractions

Semantic constraints are generated using a machine learning algorithm that learns from the solutions provided for each problem. It analysing pairs of solutions to identify similarities and differences between them. Section 3.4 provides more details on the semantic constraint generation algorithm.

The generated domain model is validated during the penultimate phase of authoring the domain model. The author requests the system to identify errors in an

incorrect solution. If errors are identified incorrectly, further example problems and solutions have to be provided by the domain expert. The author may also examine a high-level description of each generated constraint and dispute them by providing counter examples.

Finally, the domain model is deployed as a tutoring system during the final phase of the authoring process. A new instance of a tutoring system is started in ASPIRE-Tutor, which can be tested by the domain expert and made available to students. The domain expert can evaluate the effectiveness of the domain model by analysing the learning curves for constraints produced by ASPIRE-Tutor.

3.2 Architecture

ASPIRE consists of an authoring server (ASPIRE-Author) for assisting with the development of new systems, and a tutoring server (ASPIRE-Tutor) for delivering tutors. Both servers are implemented in Allegro Common Lisp [3] as web servers for users to interact through a standard web browser. All required domain-dependent information, such as the domain model and other configuration details produced by ASPIRE-Author, are transferred to ASPIRE-Tutor as an XML database.

3.2.1 Authoring Server

The authoring server consists of a set of modules, where each module is assigned a specific set of responsibilities in generating constraint-based tutors. The basic architecture of the ASPIRE-Author, as depicted in Figure 6, consists of a web interface, authoring controller, constraint generator, constraint validator and the domain model manager [8]. The domain expert interacts with each component of the web interface to generate the domain model.

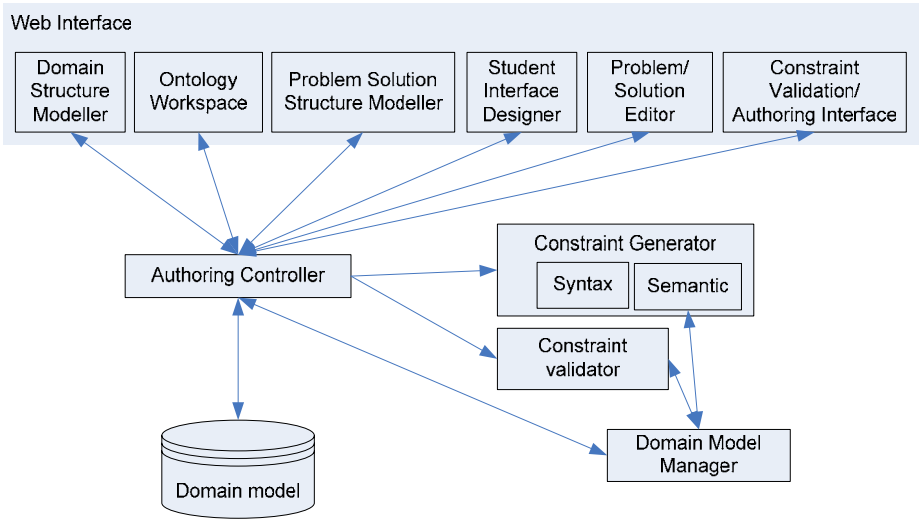

Figure 6. The architecture of ASPIRE-Author

The Authoring Controller manages the process and guides the author. This module receives all requests from the interface layer, initiates processes within other modules and returns the results to the relevant interface component.

The Syntax Constraint Generator is responsible for generating syntax constraints by analysing the domain ontology. Semantic constraints are generated by the Semantic Constraint Generator using a machine learning algorithm that learns from problems and their solutions. The Constraint Validator is responsible for carrying out all the necessary operations required for validating the constraints generated by the constraint generators.

The Domain Model Manager contains the necessary classes for storing the components of domain models. It is responsible for creating and updating domain model components such as ontology, problem solution structure, problems, solutions etc. The Domain Model Manager is also capable of producing XML representations of all domain model components for data transfer.

3.2.2 Tutoring Server

ASPIRE-Tutor (Figure 7) is also designed as a collection of modules, based on the typical ITS architecture. ASPIRE-Tutor is capable of serving a collection of tutoring systems in parallel. Each tutoring system served by ASPIRE-Tutor would have its own unique URL. Students can access the tutoring system relevant to them by pointing their browser to the appropriate URL.

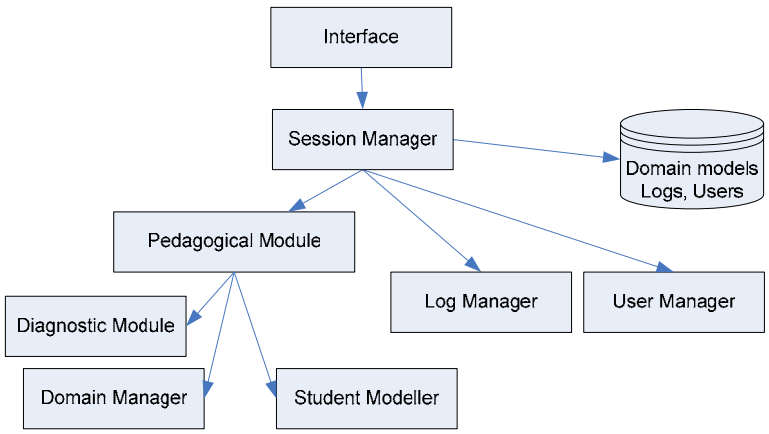

Figure 7. The architecture of ASPIRE-Tutor

The interface module is responsible for producing an interface for each tutoring system deployed on the server. The interface provides features such as login/logout, select/change problem, submit solution for evaluation etc.

The session manager is responsible for maintaining the state of each student during their interaction. The current state of a student is described by information such as the selected domain, sub-domain and problem number. The session manager also acts as the main entry point to the system, invoking the relevant modules to carry out necessary tasks. For example, when a student submits a solution to be validated,

the session manager passes on all information to the pedagogical module, which returns the feedback to be presented to the student.

The Pedagogical Module (PM) decides how to respond to each student request. It is responsible for handing all pedagogy-related requests including selecting a new problem, evaluating a student's submission and viewing the student model. In the event of evaluating a student's submission and providing feedback, the PM delegates the task of evaluating the solution to the diagnostic module and decides on the appropriate feedback by consulting the student model. The student modeler maintains a long term model of the student's knowledge.

3.3 Syntax Constraints Generation

An ontology contains a lot of information about the syntax of the domain. Composing a domain ontology is a much easier task for the author than composing constraints that check whether the student has used correct syntax. The goal of syntax constraint generator is to extract all useful syntactic information from the ontology and translate them into syntax constraints for the domain model.

Syntax constraints are generated by analysing relationships between concepts and properties of concepts specified in the ontology. The algorithm extracts the restrictions specified for relationships and properties and generates syntax constraints by translating them into constraints. These constraints are applicable to both procedural and non-procedural domains. An extra set of constraints are generated for procedural domains to ensure that the student adheres to the correct problem-solving procedure. These constraints are generated by analysing the solution structure modelled during stage three of the authoring process. The syntax constraints generation algorithm is detailed in further in [12, 14].

ASPIRE-Author produced 11 constraints for fraction addition from the ontology in Figure 3 and the solution structure in Figure 4. For example, constraint 7 is relevant while the student is carrying out the first problem solving step ('Find LCD') and its satisfaction condition ensures that the student has entered the answer. As the domain does not contain any complicated syntax restrictions, and inputs are restricted by the student interface, the generated constraints are sufficient to ensure that students use the correct syntax and the correct problem-solving procedure.

The syntax constraint generation algorithm has been evaluated in a number of domains. The evaluations carried out for the domains of ER modelling and database normalisation produced promising results. All syntax constraints that were hand-crafted in KERMIT [7, 11], a successful constraint-based tutor for ER modelling were generated by ASPIRE. Furthermore, the algorithm produced all but two syntax constraints that existed in NORMIT [6, 7], an effective tutoring system for database normalisation.

3.4 Semantic Constraints Generation

Semantic constraints ensure that a student's solution satisfies all semantic requirements of a problem, by comparing the student's and ideal solution. They are generated by a machine learning algorithm. Problems and solutions provided by the author are used as examples for semantic constraint generation. Multiple solutions for

a problem depict different ways of solving it, and enable the algorithm to generate constraints that can identify all correct solutions, regardless of the student's approach.

The algorithm generates new semantic constraints by analysing a pair of correct solutions for the same problem. Constraints are generated by identifying similarities and differences between two solutions. The process of generating constraints is iterated until all pairs of solutions are analysed. Each new pair of solutions can lead to either generalising or specialising previously generated constraints. If a newly analysed pair of solutions violate a previously generated constraint, its satisfaction condition is generalised in order to satisfy the solutions, or the constraint's relevance condition is specialised for the constraint to be irrelevant for the solutions. This algorithm is discussed in [12]. Evaluations performed show that the semantic constraints generator produced 85% of the semantic constraints found in KERMIT. Moreover, the generated constraints for the domain of database normalisation covered all the semantic constraints that exist in NORMIT.

39 semantic constraints were generated for fraction addition, from only two example problems. As each problem in this domain has only a single valid solution, semantic constraints check that the student's solution matches the ideal solution. For example, constraint 1 ensures that if the student is currently doing the first problem solving step ('Find LCD'), the LCD component of their solution is not empty (i.e., the student has specified the LCD) and the ideal solution contains an LCD (i.e. it is necessary to find the LCD for the current problem), then the student's answer needs to be equal to the one specified in the ideal solution.

The majority of generated semantic constraints ensure that relationships, such as fractions having a numerator and a denominator, exist in student solutions. As the interface implicitly forces these relationships, some semantic constraints are trivially satisfied. However, we believe that it is still necessary for the domain model to contain such constraints, because the author may design a less restrictive interface. Only two example problems were needed to generate semantic constraints for fraction addition, as the domain is very simple.

4 Conclusions

We provided an overview of ASPIRE, an authoring system that assists domain experts in building constraint-based ITSs and serves the developed tutoring systems over the web. ASPIRE follows a semi-automated process for generating domain models, and produces a fully functional web-based ITS, which can be used by students. We also outlined the constraint generation algorithms, which produced promising results during preliminary evaluations. ASPIRE-Author produced a satisfactory domain model for fraction addition, consisting of 11 syntax and 39 semantic constraints. The generated domain model can be used to power a tutoring system for students with minor modifications.

ASPIRE will be completed in July 2006, and then we will conduct a thorough evaluation of the system's effectiveness. We also intend to develop a tutorial outlining the authoring process to assist novices in building constraint-based tutoring systems using ASPIRE, especially modelling domain ontologies.

Acknowledgements: The ASPIRE project is supported by the eCDF grant from the Tertiary Education Commission of New Zealand. We thank all members of ICTG for their support.

References

1. Jarvis, M., Nuzzo-Jones, G., Heffernan, N., Applying Machine Learning Techniques to Rule Generation in Intelligent Tutoring Systems. In *ITS 2004*, (Maceio, Brazil, 2004), Springer, 541-553.
2. Koedinger, K., Aleven, V., Heffernan, N., McLaren, B. and Hockenberry, M., Openning the Door to Non-programmers: Authoring Intelligent Tutor Behavior by Demonstration. In *ITS 2004*, (Maceio, Brazil, 2004), Springer, 162-174.
3. Allegro Common Lisp (www.franz.com)
4. Martin, B., Mitrovic, A. Authoring Web-Based Tutoring Systems with WETAS. Kinshuk, R. Lewis, K. Akahori, R. Kemp, T. Okamoto, L. Henderson, C-H Lee (eds) Proc. ICCE 2002 (Auckland, 2002), 183-187.
5. Martin, B., Mitrovic, A. Domain Modelling: Art or Science? In: U. Hoppe, F. Verdejo & J. Kay (ed) Artificial Intelligence in Education 2003, 183-190.
6. Mitrovic, A. The Effect of Explaining on Learning: a Case Study with a Data Normalization Tutor. In: C-K Looi, G. McCalla, B. Bredeweg, J. Breuker (eds) Proc. Artificial Intelligence in Education, 2005, IOS Press, 499-506.
7. Mitrovic, A., Suraweera, P., Martin, B., Weerasinghe, A. DB-suite: Experiences with Three Intelligent, Web-based Database Tutors. *Journal of Interactive Learning Research*, 15, 2004, 409-432.
8. Mitrovic, A., Martin, B., Suraweera, P., Zakharov, K., Milik, N., Holland, J. ASPIRE: Functional Specification and Architectural Design**.** Tech. Report TR-COSC 05/05, University of Canterbury, 2005.
9. Mitrovic, A., Martin, B., Suraweera, P., Zakharov, K., Milik, N., Holland, J. ASPIRE: Student Modelling and Domain Specification. Tech. Report TR-COSC 08/05, University of Canterbury, 2005.
10. Murray, T. An Overview of Intelligent Tutoring System Authoring Tools: Updated analysis of the state of the art. *Authoring tools for advanced technology learning environments*. 2003, 491-545.
11. Suraweera, P., Mitrovic, A., An Intelligent Tutoring System for Entity Relationship Modelling. *Artificial Intelligence in Education*, 14, (2004), 375-417.
12. Suraweera, P., Mitrovic, A., Martin, B., A Knowledge Acquisition System for Constraint-based Intelligent Tutoring Systems. In: C-K Looi, G. McCalla, B. Bredeweg, J. Breuker (eds) Artificial Intelligence in Education, 2005, IOS Press, 638-645.
13. Suraweera, P., Mitrovic, A., Martin, B., The role of domain ontology in knowledge acquisition for ITSs. In *Intelligent Tutoring Systems 2004*, (Maceio, Brazil, 2004), Springer, 207-216.
14. Suraweera, P., Mitrovic, A., Martin, B., The use of ontologies in ITS domain knowledge authoring. in *2nd Int. Workshop on Applications of Semantic Web for E-learning SWEL'04, ITS2004*, (Maceio, Brazil, 2004), 41-49.
15. Tecuci, G. *Building Intelligent Agents: An Apprenticeship Multistrategy Learning Theory, Methodology, Tool and Case Studies*. Academic press, 1998.
16. Tecuci, G., Keeling, H. Developing an Intelligent Educational Agent with Disciple. *Artificial Intelligence in Education*, 10, 1999, 221-237.